




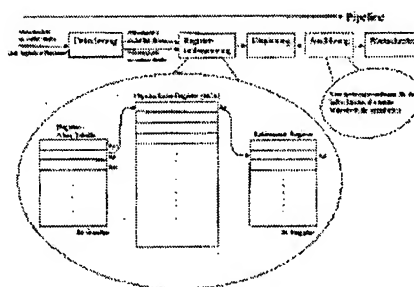


**Processor for execution of instructions of packed data****Publication number:** DE19914617**Publication date:** 1999-10-14**Inventor:** ROUSSEL PATRICE (US); THAKKAR TICKY (US)**Applicant:** INTEL CORP (US)**Classification:****- International:** G06F9/30; G06F9/302; G06F9/318; G06F9/30;  
G06F9/302; G06F9/318; (IPC1-7): G06F9/28; G06F9/38**- European:** G06F9/30R4; G06F9/30R4S; G06F9/30T; G06F9/302;  
G06F9/318; G06F9/318T**Application number:** DE19991014617 19990331**Priority number(s):** US19980053127 19980331**Also published as:** US6970994 (B2)  
 US6230253 (B1)  
 US2002010847 (A1)  
 GB2339040 (A)  
 CN1595390 (A)

more &gt;&gt;

**Report a data error here****Abstract of DE19914617**

The system has a processor with a number of registers that receive data from a decoder that receive micro instructions via a register identification unit. The data is received in a packed form and the decoder is configured such that a specific register in the system is identified. An Independent claim is included for a method for carrying out instructions in a processor.



Data supplied from the esp@cenet database - Worldwide

18 BUNDESREPUBLIK  
DEUTSCHLAND



DEUTSCHES  
PATENT- UND  
MARKENAMT

12 Offenlegungsschrift  
10 DE 199 14 617 A 1

61 Int. Cl.<sup>6</sup>:  
G 06 F 9/28  
G 06 F 9/38

21 Aktenzeichen: 199 14 617.9  
22 Anmeldetag: 31. 3. 99  
43 Offenlegungstag: 14. 10. 99

DE 199 14 617 A 1

30 Unionspriorität:  
09/053,127 31. 03. 98 US  
71 Anmelder:  
Intel Corp., Santa Clara, Calif., US  
74 Vertreter:  
Zenz, Helber, Hosbach & Partner, 45128 Essen

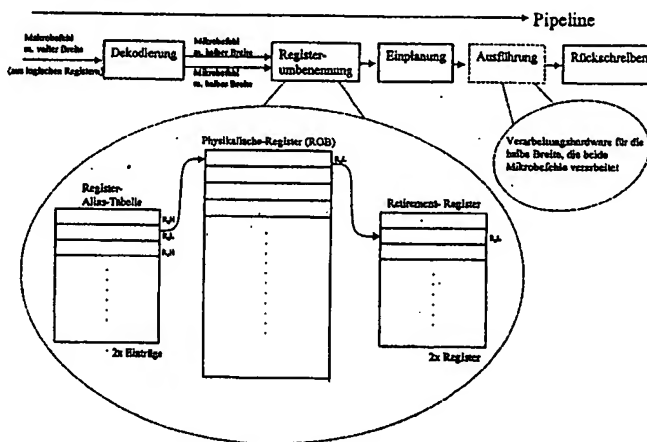
72 Erfinder:  
Roussel, Patrice, Portland, Oreg., US; Thakkar,  
Ticky, Portland, Oreg., US

Die folgenden Angaben sind den vom Anmelder eingereichten Unterlagen entnommen

Prüfungsantrag gem. § 44 PatG ist gestellt

54 Prozessor und Verfahren zum Ausführen von Befehlen an gepackten Daten

57 Es werden ein Verfahren und eine Vorrichtung zum Ausführen von skalarer Gepackte-Daten-Befehle zur Verfügung gestellt. Gemäß einem Aspekt der Erfindung umfaßt ein Prozessor eine Mehrzahl von Registern, eine mit der Mehrzahl von Registern gekoppelte Registerumbenennungseinheit, einen mit der Registerumbenennungseinheit gekoppelten Decodierer und eine mit dem Decodierer gekoppelte Ausführungseinheit partieller Breite. Die Registerumbenennungseinheit stellt eine Architekturregisterdatei zur Verfügung, um gepackte Datenoperanden zu speichern, die jeweils eine Mehrzahl von Datenelementen enthalten. Der Decodierer ist so konfiguriert, daß er einen ersten und einen zweiten Befehlssatz decodiert, wobei jeder Befehl wenigstens ein Register in der Architekturregisterdatei spezifiziert. Jeder der Befehle des ersten Befehlssatzes spezifiziert Operationen, die an sämtlichen Datenelementen ausgeführt werden sollen. Im Unterschied dazu spezifiziert jeder der Befehle des zweiten Befehlssatzes Operationen, die nur an einer Untermenge der in dem wenigstens einen spezifizierten Register gespeicherten Datenelemente durchgeführt werden sollen. Die Ausführungseinheit partieller Breite ist so konfiguriert, daß sie sowohl von dem ersten als auch von dem zweiten Befehlssatz spezifizierte Operationen ausführt.



DE 199 14 617 A 1

Die Erfindung bezieht sich auf einen Prozessor mit einer Mehrzahl von Registern und einer Registerumbenennungseinheit, die mit der Mehrzahl von Registern gekoppelt ist, um eine Architekturregisterdatei zum Speichern gepackter Datenoperanden zur Verfügung zu stellen, wobei jeder der gepackten Datenoperanden eine Mehrzahl von Datenelementen aufweist. Ferner bezieht sich die Erfindung auf ein Verfahren zum Ausführen eines Makrobefehls in einem solchen Prozessor. Dabei verwendet der Prozessor beispielsweise die SIMD-Technologie.

Multimedia-Anwendungen, wie beispielsweise 2D/3D-Grafikanwendungen, Bildverarbeitung, Videokompression/Dekompression, Spracherkennungsalgorithmen und Audioverarbeitungen erfordern oftmals, daß die gleiche Operation an einer großen Anzahl von Datenelementen ausgeführt werden soll (was als "Datenparallelität") bezeichnet wird. Jede Art von Multimediaanwendungen implementiert typischerweise einen oder mehrere Algorithmen, die eine Anzahl von Gleitkomma- oder Ganzzahloperationen, wie beispielsweise Addieren (ADD) oder Multiplizieren (MUL) erfordern. Indem Makrobefehle zur Verfügung gestellt werden, deren Ausführung einen Prozessor veranlaßt, die gleiche Operation an mehreren Datenelementen parallel auszuführen (Einzelbefehl-Mehrfachdaten (SIMD)-Technologie, wie sie beispielsweise von der Pentium®-Prozessor-Architektur und dem MMX-Befehlssatz verwendet wird), wurde eine signifikante Verbesserung bei der Leistung von Multimedia-Anwendungen ermöglicht (Pentium® und MMX™ sind eingetragene Marken beziehungsweise Handelsbezeichnungen der Intel Corporation in Santa Clara, Kalifornien). Die SIMD-Technologie ist insbesondere für solche Systeme geeignet, die gepackte Datenformate vorsehen. Ein gepacktes Datenformat ist ein solches, bei dem die Bits in einem Register logisch in eine Anzahl von Datenelementen fester Größe unterteilt sind, wobei jedes Datenelement einen separaten Wert repräsentiert. Beispielsweise kann ein 64-Bit-Register in vier 16-Bit-Elemente aufgeteilt werden, von denen jedes einen separaten 16-Bit-Wert repräsentiert. Gepackte-Daten-Befehle können dann separat jedes Element in diesen gepackten Daten parallel verarbeiten.

Es wird auf Fig. 1 Bezug genommen, in der ein Beispiel eines Gepackte-Daten-Befehls veranschaulicht ist. Bei diesem Beispiel addiert ein gepackter ADD-Befehl (beispielsweise ein SIMD-ADD-Befehl) zugehörige Datenelemente eines ersten gepackten Datenoperanden X und eines zweiten gepackten Datenoperanden Y, um ein Ergebnis Z gepackter Daten zu erzeugen, d. h.  $X_0+Y_0=Z_0$ ,  $X_1+Y_1=Z_1$ ,  $X_2+Y_2=Z_2$  und  $X_3+Y_3=Z_3$ . Das Packen vieler Datenelemente in ein Register bzw. einen Speicherplatz und das Benutzen einer parallelen Hardware-Ausführung gestattet es der SIMD-Architektur, mehrere Operationen gleichzeitig durchzuführen, was zu einer signifikanten Leistungsverbesserung führt. Beispielsweise können bei diesem Beispiel vier einzelne Ergebnisse in derselben Zeit erlangt werden, die zuvor erforderlich war, um ein einziges Ergebnis zu gewinnen.

Während die durch die SIMD-Architektur erreichten Vorteile offensichtlich sind, verbleiben Situationen, in welchen es wünschenswert ist, einzelne Ergebnisse für nur eine Untermenge der gepackten Datenelemente zurückzugeben. Darin besteht die der Erfindung zugrundeliegende Aufgabe.

Diese Aufgabe wird erfindungsgemäß durch einen Prozessor mit den Merkmalen des Patentanspruchs 1 bzw. ein Verfahren mit den Merkmalen des Patentanspruchs 6 gelöst.

Es werden ein Verfahren und eine Vorrichtung (Prozessor) zum Ausführen von Gepackte-Daten-Befehlen für eine partielle Breite angegeben. Gemäß einem Aspekt der Erfin-

dung enthält ein Prozessor eine Mehrzahl von Registern, eine mit der Mehrzahl von Registern gekoppelte Registerumbenennungseinheit, einen mit der Registerumbenennungseinheit gekoppelten Decodierer und eine mit dem Decodierer gekoppelte Ausführungseinheit partieller Breite. Die Registerumbenennungseinheit stellt eine Architekturregisterdatei zur Verfügung, um gepackte Datenoperanden zu speichern, von denen jeder eine Mehrzahl von Datenelementen enthält. Der Decodierer ist so konfiguriert, daß er einen ersten und einen zweiten Befehlssatz decodieren kann, wobei jeder Befehl wenigstens ein Register in der Architekturregisterdatei angibt. Jeder der Befehle des ersten Befehlssatzes spezifiziert Operationen, die an sämtlichen der in dem wenigstens einen spezifizierten Register gespeicherten Datenelemente ausgeführt werden sollen. Im Unterschied dazu spezifiziert jeder der Befehle des zweiten Befehlssatzes Operationen, die nur an einer Untermenge der in dem wenigstens einen spezifizierten Register gespeicherten Datenelemente ausgeführt werden sollen. Die Ausführungseinheit partieller Breite ist so konfiguriert, daß sie von sowohl dem ersten als auch von dem zweiten Satz von Befehlen spezifizierte Operationen ausführen kann.

Vorteilhafte Weiterbildungen der Erfindung sind in den Unteransprüchen gekennzeichnet.

Im folgenden wird die Erfindung anhand von in der Zeichnung beschriebenen Ausführungsbeispielen näher erläutert. In der Zeichnung zeigen:

Fig. 1 einen gepackten ADD-Befehl, der zugehörige Datenelemente aus einem ersten gepackten Datenoperanden und einem zweiten gepackten Datenoperanden addiert.

Fig. 2A ist eine vereinfachte Blockdarstellung, die ein beispielhaftes Computersystem nach einem Ausführungsbeispiel der Erfindung veranschaulicht.

Fig. 2B ist eine vereinfachte Blockdarstellung, die einen Beispielsatz logischer Register gemäß einem Ausführungsbeispiel der Erfindung veranschaulicht.

Fig. 2C ist eine vereinfachte Blockdarstellung, die Beispielsätze logischer Register gemäß einem weiteren Ausführungsbeispiel der Erfindung veranschaulicht.

Fig. 3 ist ein Ablaufdiagramm, das die Befehlsausführung nach einem Ausführungsbeispiel der Erfindung veranschaulicht.

Fig. 4 veranschaulicht das Ergebnis der Ausführung eines Befehls für eine partielle Breite gepackter Daten gemäß den verschiedenen Ausführungsbeispielen der Erfindung.

Fig. 5A veranschaulicht eine Schaltungsanordnung zum Ausführen eines Befehls für gepackte Daten voller Breite und für gepackte Daten partieller Breite gemäß einem Ausführungsbeispiel der Erfindung.

Fig. 5B veranschaulicht eine Schaltungsanordnung zum Ausführen von Befehlen an gepackten Daten voller Breite und partieller Breite gemäß einem weiteren Ausführungsbeispiel der Erfindung.

Fig. 5C veranschaulicht eine Schaltungsanordnung zum Ausführen von Befehlen an gepackten Daten voller Breite und partieller Breite gemäß einem weiteren Ausführungsbeispiel der Erfindung.

Fig. 6 veranschaulicht eine ADD-Ausführungseinheit und eine MUL-Ausführungseinheit, die in der Lage sind, als vier separate ADD-Ausführungseinheiten bzw. vier separate MUL-Ausführungseinheiten zu arbeiten, gemäß einer beispielhaften Prozessorimplementierung von SIMD.

Fig. 7A-7B veranschaulichen eine Operation an gepackten Daten voller Breite und eine Operation an gepackten Daten partieller Breite, die in einer "gestaffelten" Weise ausgeführt werden.

Fig. 8A veranschaulicht eine Schaltungsanordnung innerhalb eines Prozessors, die auf Operanden voller Breite aus

logischen Registern zugreift, während sie jeweils Operationen an der halben Breite der Operanden ausführt.

Fig. 8B ist ein Zeitdiagramm, das die Schaltungsanordnung gemäß Fig. 8A weiter veranschaulicht.

Fig. 9 veranschaulicht ein Ausführungsbeispiel einer Auker-Reihe-Pipeline zum Ausführen von Operationen an Operanden in einer "gestaffelten" Weise, indem ein Makrobefehl in eine Mehrzahl von Mikrobefehlen konvertiert wird, die jeweils einen Abschnitt des Operanden voller Breite bearbeiten.

Fig. 10 ist ein Zeitschaubild, das das in Fig. 9 beschriebene Ausführungsbeispiel weiter veranschaulicht.

Fig. 11 ist ein Blockschaubild, das die Dekodierlogik veranschaulicht, die benutzt werden kann, um die decodierende Verarbeitung gemäß einem Ausführungsbeispiel der Erfindung durchzuführen.

Es werden ein Verfahren und eine Vorrichtung zum Ausführen von Befehlen an gepackten Daten partieller Breite beschrieben. Hierbei wird die Bezeichnung "Befehl an gepackten Daten voller Breite" verwendet, um einen Befehl an gepackten Daten (z. B. einen SIMD-Befehl) zu bezeichnen, der an sämtlichen Datenelementen wenigstens eines gepackten Datenoperanden ausgeführt wird. Im Unterschied dazu bezeichnet der Begriff "Befehl an gepackten Daten partieller Breite" ganz allgemein einen Befehl an gepackten Daten, der nur eine Untermenge der Datenelemente des wenigstens einen gepackten Datenoperanden verarbeiten soll und der ein gepacktes Datenergebnis (beispielsweise an eine Registerdatei gepackter Daten) zurückgibt. Beispielsweise kann ein skalarer SIMD-Befehl nur ein Ergebnis einer Operation zwischen dem am geringsten bewerteten Paar von Datenelementen der gepackten Datenoperanden erfordern. Bei diesem Beispiel können die verbleibenden Datenelemente des gepackten Datenergebnisses unberücksichtigt bleiben, da sie keinen Einfluß auf den skalaren SIMD-Befehl haben (beispielsweise können die verbleibenden Datenelemente beliebigen Inhalts sein). Gemäß den verschiedenen Ausführungsbeispielen der Erfindung können Ausführungseinheiten so konfiguriert werden, daß sie effizient sowohl Befehle an gepackten Daten voller Breite (z. B. SIMD-Befehle) als auch einen Satz von Befehlen an gepackten Daten partieller Breite (z. B. skalare SIMD-Befehle) unterbringen können.

In der folgenden detaillierten Beschreibung werden aus Gründen der Erläuterung zahlreiche spezielle Details angegeben, um ein besseres Verständnis der Erfindung zu erreichen. Für einen Fachmann ist es jedoch klar, daß diese speziellen Details nicht verwendet werden müssen, um die Erfindung auszuführen. An anderen Stellen werden gut bekannte Einrichtungen, Strukturen, Schnittstellen und Prozesse nicht oder nur in Blockdarstellung gezeigt.

#### Berechtigung von Befehlen an gepackten Daten partieller Breite

Betrachtet man den Umfang der Software, die für skalare Architekturen (z. B. Einzelbefehl-Einzeldaten(SISD)-Architekturen), die skalare Operationen an Gleitkommatdaten einfacher Genauigkeit, Gleitkommatdaten doppelter Genauigkeit und Ganzzahldaten benutzen, geschrieben worden ist, so ist es wünschenswert, Entwicklern die Option zur Verfügung zu stellen, ihre Software auf Architekturen zu portieren, die Befehle an gepackten Daten unterstützt, wie beispielsweise SIMD-Architekturen, ohne daß deren Software umgeschrieben werden muß und/oder neue Befehle gelernt werden müssen. Durch die Schaffung von Befehlen an gepackten Daten partieller Breite kann eine einfache Übersetzung alten skalaren Code in skalaren gepackten Datencode transformieren. Beispielsweise wäre es sehr leicht für einen

Compiler, skalare SIMD-Befehle aus skalarem Code zu erzeugen. Wenn dann die Entwickler erkennen, daß Abschnitte ihrer Software unter Verwendung von SIMD-Befehlen optimiert werden können, können sie schrittweise Vorteil aus den Befehlen für gepackte Daten ziehen. Selbstverständlich ist es wahrscheinlich, daß die SIMD-Technologie benutzende Computersysteme außerdem abwärtskompatibel bleiben, indem sie SISD-Befehle ebenso unterstützen. Jedoch machen es viele jüngere Architekturverbesserungen und andere hier erörterte Faktoren vorteilhaft für Entwickler, zur SIMD-Technologie überzugehen und sie auszunutzen, selbst wenn zunächst nur skalare SIMD-Befehle verwendet werden.

Eine weitere Rechtfertigung für das Vorsehen von Befehlen für gepackte Daten partieller Breite besteht in den vielen Vorteilen, die erreicht werden können, indem nur eine Untermenge des Operanden voller Breite bearbeitet wird, einschließlich des reduzierten Stromverbrauchs, der erhöhten Geschwindigkeit, eines sauberen Ausnahmmodells und einer erhöhten Speicherung. Wie unten erläutert wird, können auf der Grundlage eines mit dem Befehl für gepackte Daten partieller Breite zur Verfügung gestellten Hinweises Stromersparungen erreicht werden, indem selektiv diejenigen Hardwareeinheiten abgeschaltet werden, die zum Ausführen der gegenwärtigen Operation nicht erforderlich sind.

Eine weitere Situation, in welcher es nicht erwünscht ist, einen Befehl für gepackte Daten zu zwingen, einzelne Ergebnisse für jedes Paar von Datenelementen zurückzugeben, umfaßt arithmetische Operationen in einer Umgebung, die eine Hardware partieller Breite zur Verfügung stellt. Infolge der Kosten- und/oder Chip-Einschränkungen ist es üblich, für bestimmte arithmetische Operationen, wie beispielsweise das Dividieren, nicht die vollständige Unterstützung zur Verfügung zu stellen. Ihrer Natur nach ist die Divisionsoperation sehr lang, selbst wenn Hardware voller Breite (beispielsweise eine Eins-zu-eins-Korrespondenz zwischen Ausführungseinheiten und Datenelementen) implementiert ist. Folglich wird in einer Umgebung, die nur Operationen an gepackten Daten voller Breite unterstützt, während Hardware partieller Breite zur Verfügung gestellt wird, die Verzögerung noch länger. Wie unten näher veranschaulicht wird, kann eine Operation für gepackte Daten partieller Breite, wie beispielsweise eine Divisionsoperation für gepackte Daten partieller Breite, es selektiv bestimmten Abschnitten ihrer Operanden gestatten, die Divisionshardware zu umgehen. Auf diese Weise wird kein Leistungsnachteil dadurch verursacht, daß nur eine Untermenge der Datenelemente in den gepackten Datenoperanden bearbeitet wird.

Zusätzlich können Ausnahmen, die in Verbindung mit unwesentlichen Datenelementen entstehen, eine Konfusion für den Entwickler und/oder eine Inkompatibilität zwischen SISD- und SIMD-Maschinen verursachen. Folglich ist es vorteilhaft, Ausnahmen nur für diejenigen Datenelemente zu berichten, für die der Befehl gedacht ist. Eine Unterstützung von Befehlen für gepackte Daten partieller Breite gestattet, daß ein vorhersagbares Ausnahmmodell erreicht wird, indem das Auslösen von Ausnahmebedingungen auf diejenigen begrenzt wird, die in Verbindung mit solchen Datenelementen auftreten, die bearbeitet werden, oder bei welchen durch unwesentliche Datenelemente erzeugte Ausnahmen wahrscheinlich eine Konfusion oder Inkompatibilität zwischen SISD- und SIMD-Maschinen verursachen würde.

Schließlich stellen bei Ausführungsbeispielen, bei denen Abschnitte des gepackten Zieldatenoperanden nicht im Ergebnis der Ausführung einer Operation für gepackte Daten partieller Breite verfälscht werden, Befehle für gepackte Daten partieller Breite effektiv einen zusätzlichen Register-

raum zum Speichern von Daten zur Verfügung. Wenn beispielsweise an dem unteren Abschnitt des gepackten Datenoperanden eine Operation ausgeführt wird, können Daten im oberen Abschnitt gespeichert werden und umgekehrt.

#### Ein Beispielcomputersystem

Fig. 2A ist ein vereinfachtes Blockschaltbild, das ein Beispielcomputersystem gemäß einem Ausführungsbeispiel der Erfindung veranschaulicht. Bei dem dargestellten Ausführungsbeispiel enthält ein Computersystem 200 einen Prozessor 205, eine Speichereinrichtung 210 und einen Bus 215. Der Prozessor 205 ist mit der Speichereinrichtung 210 durch den Bus 215 gekoppelt. Zusätzlich sind eine Reihe von Benutzereingabe/Ausgabeeinrichtungen, wie beispielsweise eine Tastatur 220 und eine Anzeige 225, ebenfalls mit dem Bus 215 gekoppelt. Das Computersystem 200 kann darüber hinaus mit einem Netzwerk 230 über den Bus 215 gekoppelt sein. Der Prozessor 205 repräsentiert eine zentrale Verarbeitungseinheit einer beliebigen Architektur, wie beispielsweise einer CISC-, RISC-, VLIW- oder Hybrid-Architektur. Darüber hinaus kann der Prozessor 205 auf einem oder mehreren Chips implementiert sein. Die Speichereinrichtung 210 repräsentiert einen oder mehrere Mechanismen zum Speichern von Daten. Beispielsweise kann die Speichereinrichtung 210 einen Nur-Lese-Speicher (ROM), einen Speicher mit wahlfreiem Zugriff (RAM), ein magnetisches Plattenspeichermedium, ein optisches Speichermedium, Flash-Speicherbauelemente und/oder andere maschinenlesbare Medien umfassen. Der Bus 215 repräsentiert einen oder mehrere Busse (z. B. AGP, PCI, ISA, X-Bus, EISA, VESA usw.) und Brücken (die auch als Bussteuereinrichtungen bezeichnet werden). Während dieses Ausführungsbeispiel in Bezug auf ein Einzelprozessor-Computersystem beschrieben wird, ist es klar, daß die Erfindung auch in einem Multi-Prozessor-Computersystem implementiert werden kann. Außerdem ist, obwohl das vorliegende Ausführungsbeispiel in Bezug auf ein 32-Bit- und ein 64-Bit-Computersystem beschrieben wird, die Erfindung nicht auf ein derartiges Computersystem begrenzt.

Fig. 2A veranschaulicht zusätzlich, daß der Prozessor 205 eine Befehlssatzereinheit 260 enthält. Selbstverständlich enthält der Prozessor 205 zusätzliche Schaltungen; jedoch sind derartige zusätzliche Schaltungen nicht für das Verständnis der Erfindung erforderlich. Auf jeden Fall enthält die Befehlssatzereinheit 260 die Hardware und/oder Firmware zum Decodieren und Ausführen eines oder mehrerer Befehlssätze. Bei dem dargestellten Ausführungsbeispiel enthält die Befehlssatzereinheit 260 eine Dekodier-/Ausführungs-Einheit 275. Die Dekodiereinheit dekodiert Befehle, die von dem Prozessor 205 empfangen worden sind, in einen oder mehrere Mikrobefehle. Die Ausführungseinheit führt in Erweiterung der von der Dekodiereinheit empfangenen Mikrobefehle geeignete Operationen durch. Die Dekodiereinheit kann unter Verwendung einer Reihe unterschiedlicher Mechanismen (beispielsweise einer Nachschlagetabelle, einer Hardware-Implementierung, einer PLA etc.) implementiert werden.

Bei dem vorliegenden Beispiel ist eine Dekodier-/Ausführungseinheit 275 gezeigt, die einen Befehlssatz 280 enthält, der sowohl Befehle für gepackte Daten voller Breite als auch Befehle für gepackte Daten partieller Breite enthält. Diese Befehle für gepackte Daten können, wenn sie ausgeführt werden, den Prozessor 205 veranlassen, gepackte Gleitkommaoperationen voller/partieller Breite und/oder gepackte Ganzzahloperationen voller/partieller Breite durchzuführen. Zusätzlich zu den Befehlen für gepackte Daten kann der Befehlssatz 280 weitere in vorhandenen Mikro-

prozessoren zu findende Befehle enthalten. Beispielsweise unterstützt der Prozessor 205 bei einem Ausführungsbeispiel einen Befehlssatz, der mit der Intel-32-Bit-Architektur (IA-32) und/oder der Intel-64-Bit-Architektur (IA-64) kompatibel ist.

Außerdem ist in der Befehlssatzereinheit 260 eine Speichereinheit 285 enthalten. Die Speichereinheit 285 kann einen oder mehrere Sätze von Architekturregistern (auch als logische Register bezeichnet), enthalten, die von dem Prozessor 205 zum Speichern von Informationen, einschließlich Gleitkommadaten und gepackten Gleitkommadaten, benutzt werden. Zusätzlich können weitere logische Register zum Speichern von Ganzzahldaten, gepackten Ganzzahldaten und verschiedenen Steuerdaten, wie beispielsweise einer Anzeige für die Spitze des Stapelspeichers und dergleichen, enthalten sein. Die Begriffe Architekturregister und logische Register werden im folgenden verwendet, um auf das Konzept der Art und Weise Bezug zu nehmen, auf welche Befehle einen Speicherbereich angeben, der einen einzelnen Operanden enthält. So kann ein logisches Register in Hardware unter Verwendung einer beliebigen Anzahl gut bekannter Techniken implementiert sein, einschließlich eines speziellen physikalischen Registers, eines oder mehrerer dynamisch unter Verwendung eines Registerumbenennungsmechanismus (detaillierter unten beschrieben) zugewiesener physikalischer Register, etc. In jedem Fall repräsentiert ein logisches Register die kleinste Einheit eines durch einen Befehl für gepackte Daten adressierbaren Speichers.

Bei dem dargestellten Ausführungsbeispiel ist in der Speichereinrichtung 210 ein Betriebssystem (OS) 235 und eine Routine für gepackte Daten 240 zur Ausführung durch das Computersystem 200 gespeichert. Die Routine 240 für gepackte Daten ist eine Sequenz von Befehlen, die einen oder mehrere Befehle für gepackte Daten, wie beispielsweise skalare SIMD-Befehle oder SIMD-Befehle enthalten kann. Wie unten näher erörtert wird, gibt es Situationen, einschließlich Geschwindigkeits-, Stromverbrauchs- und Ausnahmebehandlung, in denen es erwünscht ist, eine Operation nur an einer Untermenge von Datenelementen in einem gepackten Datenoperanden oder einem Paar gepackter Datenoperanden durchzuführen (oder nur für diese Untermenge einzelne Ergebnisse zurückzugeben). Folglich ist es für den Prozessor 205 vorteilhaft, in der Lage zu sein, zwischen Befehlen für gepackte Daten voller Breite und Befehlen für gepackte Daten partieller Breite unterscheiden zu können und diese entsprechend ausführen zu können.

Fig. 2B ist eine vereinfachte Blockdarstellung, die Beispielsätze von logischen Registern gemäß einem Ausführungsbeispiel der Erfindung veranschaulicht. Bei diesem Beispiel enthält die Speichereinheit 285 eine Mehrzahl skalarer Gleitkommae Register 291 (eine skalare Registerdatei) und eine Mehrzahl gepackter Gleitkommae Register 292 (eine Registerdatei für gepackte Daten). Die skalaren Gleitkommae Register 291 (z. B. die Register  $R_0$ - $R_7$ ) können als eine als Stapel referenzierte Registerdatei implementiert sein, wenn Gleitkommae Befehle so ausgeführt werden, daß sie kompatibel mit der vorhandenen, für die Intel-Architektur geschriebenen Software sein sollen. Bei alternativen Ausführungsbeispielen jedoch können die Register 291 als eine flache Registerdatei behandelt werden. Bei dem dargestellten Ausführungsbeispiel ist jedes der gepackten Gleitkommae Register (z. B.  $XMM_0$ - $XMM_7$ ) als ein einzelnes logisches 128-Bit-Register implementiert. Es ist jedoch klar, daß breitere oder schmalere Register benutzt werden können, um an eine Implementierung angepaßt zu sein, die mehr oder weniger Datenelemente oder größere oder kleinere Datenelemente verwendet. Zusätzlich können mehr oder weniger gepackte Gleitkommae Register 292 vorgesehen

sein. Ähnlich wie die skalaren Gleitkommaregister 291 können die gepackten Gleitkommaregister 292 entweder als stapelreferenzierte Registerdatei oder als flache Registerdatei implementiert werden, wenn die gepackten Gleitkommabe-  
fehle ausgeführt werden

Fig. 2C ist eine vereinfachte Blockdarstellung, die beispielhafte Sätze logischer Register nach einem weiteren Ausführungsbeispiel der Erfindung veranschaulicht. Bei diesem Beispiel enthält die Speichereinheit 285 wiederum eine Mehrzahl skalarer Gleitkommaregister 291 (eine skalare Registerdatei) und eine Mehrzahl gepackter Gleitkommaregister 292 (eine Registerdatei gepackter Daten). Jedoch werden bei dem dargestellten Ausführungsbeispiel sämtliche gepackten Gleitkommaregister (z. B.  $XMM_0$ - $XMM_7$ ) als einander entsprechende Paare hoher 293 und niedriger Register 294 implementiert. Wie unten näher erörtert werden wird, ist es für die Zwecke der Befehlsdekodierung vorteilhaft, den logischen Registeradressraum für die gepackten Gleitkommaregister 292 so zu organisieren, daß sich die Paare hoher und niedriger Register durch ein einzelnes Bit unterscheiden. Beispielsweise können die hohen und niedrigen Abschnitte von  $XMM_0$ - $XMM_7$  durch das MSB (most significant bit) differenziert werden. Vorzugsweise ist jedes der gepackten Gleitkommaregister 291 breit genug, um vier 32-Bit-Gleitkommadata-Elemente einfacher Genauigkeit aufzunehmen. Wie oben jedoch können breitere oder schmalere Register benutzt werden, um an Implementierungen angepaßt zu werden, die mehr oder weniger Datenelemente bzw. größere oder kleinere Datenelemente verwenden. Zusätzlich können, während die logischen gepackten Gleitkommaregister 292 bei diesem Beispiel jeweils einander entsprechende Paare von 64-Bit-Registern aufweisen, bei alternativen Ausführungsbeispielen sämtliche gepackten Gleitkommaregister eine beliebige Anzahl von Registern umfassen.

#### Übersicht über die Befehlsausführung

Nachdem ein Beispielcomputersystem beschrieben worden ist, in welchem ein Ausführungsbeispiel der Erfindung implementiert werden kann, wird jetzt die Befehlsausführung beschrieben.

Fig. 3 ist ein Ablaufdiagramm, das die Befehlsausführung gemäß einem Ausführungsbeispiel der Erfindung veranschaulicht. Im Schritt 310 wird ein Befehl durch den Prozessor 205 empfangen. Im Schritt 320 wird die Verarbeitung auf der Grundlage der Art des Befehls (ob es ein Befehl für gepackte Daten partieller Breite (zum Beispiel ein skalarer SIMD-Befehl) oder ein Befehl für gepackte Daten voller Breite (zum Beispiel ein SIMD-Befehl) ist) entweder mit dem Schritt 330 oder dem Schritt 340 fortgesetzt. Üblicherweise wird in der Dekodiereinheit die Art des Befehls auf der Grundlage der in dem Befehl enthaltenen Informationen bestimmt. Beispielsweise können die Informationen einen Präfix oder Suffix enthalten, der an den Befehlscode angehängt oder über einen Direktdatenwert zur Verfügung gestellt wird, um anzuzeigen, ob die zugehörige Operation an sämtlichen oder einer Untermenge der Datenelemente des gepackten Datenoperanden bzw. der gepackten Datenoperanden ausgeführt werden soll. Auf diese Weise kann der gleiche Befehlscode sowohl für Operationen an gepackten Daten voller Breite als auch für Operationen an gepackten Daten partieller Breite verwendet werden. Alternativ kann ein Satz von Befehlscodes für Operationen an gepackten Daten partieller Breite und ein anderer Befehlscodesatz für Operationen an gepackten Daten voller Breite verwendet werden.

In jedem Fall wird dann, wenn der Befehl ein herkömmli-

cher Befehl für gepackte Daten voller Breite ist, im Schritt 330 ein gepacktes Datenergebnis bestimmt, indem die von dem Befehl spezifizierte Operation an jedem der Datenelemente in dem bzw. den Operanden ausgeführt wird. Wenn jedoch der Befehl ein Befehl für gepackte Daten partieller Breite ist, dann wird im Schritt 340 ein erster Abschnitt des Ergebnisses bestimmt, indem die von dem Befehl spezifizierte Operation an einer Untermenge der Datenelemente durchgeführt wird und indem der Rest des Ergebnisses auf einen oder mehrere vorgegebene Werte gesetzt wird. Bei einem Ausführungsbeispiel ist der vorgegebene Wert der Wert der jeweils einander entsprechenden Datenelemente in einem der Operanden. Das heißt, es können Datenelemente von den Datenelementen eines der Operanden zu den jeweils entsprechenden Datenelementen in dem gepackten Datenergebnis "durchgeleitet" werden. Bei einem anderen Ausführungsbeispiel können die Datenelemente in dem verbleibenden Abschnitt des Ergebnisses sämtlich gelöscht (auf Null gesetzt) werden. Eine Beispiellogik zum Durchführen des Durchleitens der Datenelemente von einem der Operanden zu dem Ergebnis und eine Beispiellogik zum Löschen der Datenelemente in dem Ergebnis werden unten beschrieben.

Fig. 4 veranschaulicht das Ergebnis der Ausführung eines Befehls für gepackte Daten partieller Breite gemäß den verschiedenen Ausführungsbeispielen der Erfindung. Bei diesem Beispiel wird eine Operation an Datenelementen von zwei logischen Quellenregistern 410 und 420 durch eine Ausführungseinheit 440 ausgeführt. Die Ausführungseinheit 440 enthält eine Schaltungsanordnung und Logik zum Durchführen der von dem Befehl spezifizierten Operationen. Zusätzlich kann die Ausführungseinheit 440 eine Auswahl-schaltung enthalten, die es der Ausführungseinheit 440 gestattet, entweder in einem Modus für gepackte Daten partieller Breite oder in einem Modus für gepackte Daten voller Breite zu arbeiten. Beispielsweise kann die Ausführungseinheit 440 eine Durchleitschaltung zum Durchleiten von Datenelementen von einem der logischen Quellregister 410, 420 zu dem logischen Zielregister 430 enthalten, oder sie kann eine Löschschaltung zum Löschen eines oder mehrerer Datenelemente des logischen Zielregisters 430 enthalten. Zahlreiche andere Techniken können ebenfalls benutzt werden, um das Ergebnis der Operation zu beeinflussen, wobei diese Techniken einschließen, daß eines der Eingangssignale für die Operation auf einen vorgegebenen Wert gezwungen wird, wie beispielsweise einen Wert, der die Operation zwingen würde, ihre Identitätsfunktion durchzuführen, oder einen Wert, der durch die arithmetischen Operationen laufen kann, ohne eine Ausnahme (z. B. eine quiet not-a-number (QNaN)) zu signalisieren.

Bei dem veranschaulichten Beispiel wird nur das Ergebnis ( $Z_0$ ) der Operation an dem ersten Paar von Datenelementen ( $X_0$  und  $Y_0$ ) in dem logischen Zielregister 430 gespeichert. Nimmt man an, daß die Ausführungseinheit 440 eine Durchleitleitlogik enthält, so werden die verbleibenden Datenelemente des logischen Zielregisters 430 auf die Werte aus den zugehörigen Datenelementen des logischen Quellregisters 410 (d. h.  $X_3$ ,  $X_2$  und  $X_1$ ) gesetzt. Während das logische Zielregister 430 als ein separates logisches Register gezeigt ist, ist es wichtig anzumerken, daß es gleichzeitig als eines der logischen Quellregister 410, 420 dienen kann. Folglich sollte es klar sein, daß das Setzen der Datenelemente des logischen Zielregisters 430 auf Werte aus einem der logischen Quellregister 410, 420 in diesem Kontext bedeuten kann, daß in diesen Datenelementen überhaupt nichts passiert. Beispielsweise können in dem Fall, daß das logische Quellregister 410 sowohl ein logisches Quell- als auch ein Zielregister ist, verschiedene Ausführungsbeispiele

Vorteile daraus ziehen, und einfach eines oder mehrere der Datenelemente, die durchgeleitet werden sollen, nicht berühren.

Alternativ kann die Ausführungseinheit 440 eine Löschlogik enthalten. So werden anstelle des Durchleitens von Werten von einem der logischen Quellregister zu dem logischen Zielregister 430 diejenigen Datenelemente in dem Ergebnis, die nicht erforderlich sind, gelöscht. Wiederum wird bei diesem Beispiel nur das Ergebnis ( $Z_0$ ) der Operation an dem ersten Paar von Datenelementen ( $X_0$  und  $Y_0$ ) in dem logischen Zielregister 430 gespeichert. Die verbleibenden Datenelemente des logischen Zielregisters 430 werden "gelöscht" (d. h. auf Null oder auf einen anderen für diesen Fall vorgegebenen Wert gesetzt).

#### Hardware voller Breite

Die Fig. 5A–5C veranschaulichen Ausführungseinheiten 540, 560 bzw. 580, welche sowohl Befehle für gepackte Daten voller Breite als auch für gepackte Daten partieller Breite ausführen können. Die in den Ausführungseinheiten gemäß Fig. 5A und 5C enthaltene Auswahllogik stellt eine beispielhafte Durchleitlogik dar, während die Auswahllogik gemäß Fig. 5B für eine Löschlogik, die benutzt werden kann, repräsentativ ist. Bei den dargestellten Ausführungsbeispielen enthalten die Ausführungseinheiten 540, 560 und 580 jeweils eine geeignete Logik, Schaltungsanordnung und/oder Firmware zum gleichzeitigen Ausführen einer Operation 570, 571 und 572 an den Operanden voller Breite ( $X$  und  $Y$ ).

Es wird jetzt auf Fig. 5A Bezug genommen, bei der die Ausführungseinheit 540 eine Auswahllogik (zum Beispiel Multiplexer (MUX) 555–557) enthält, um zwischen einem durch die Operation 570 erzeugten Wert und einem aus einem entsprechenden Datenelement eines der Operanden herrührenden Wert auszuwählen. Die MUX 555–557 können beispielsweise durch ein Signal gesteuert werden, das anzeigt, ob die gerade ausgeführte Operation eine Operation an gepackten Daten voller Breite oder eine Operation an gepackten Daten partieller Breite ist. Bei alternativen Ausführungsbeispielen kann eine zusätzliche Flexibilität dadurch erreicht werden, daß eine zusätzliche MUX für das Datenelement 0 eingeschlossen wird und/oder jede MUX unabhängig gesteuert wird. Verschiedene Einrichtungen zum Schaffen einer MUX-Steuerung sind möglich. Gemäß einem Ausführungsbeispiel kann eine solche Steuerung aus dem Befehl selbst herrühren, oder aus ihm abgeleitet werden oder sie kann durch einen Direktwert zur Verfügung gestellt werden. Beispielsweise kann ein 4-Bit-Direktwert, der dem Befehl zugeordnet ist, verwendet werden, um eine direkte Steuerung der MUX 555–557 durch Software zu gestatten. Diejenigen Multiplexer, die zu einer Eins in dem Direktwert gehören, können angewiesen werden, das Ergebnis der Operation auszuwählen, während diejenigen Multiplexer, die zu einer Null gehören, veranlaßt werden können, das Durchleiten von Daten auszuwählen. Selbstverständlich kann eine größere oder geringere Auflösung bei verschiedenen Implementierungen erreicht werden, indem mehr oder weniger Bits zum Darstellen des Direktwerts benutzt werden.

Es wird jetzt auf Fig. 5B Bezug genommen, bei der die Ausführungseinheit 540 eine Auswahllogik (z. B. Multiplexer 565–567) zum Auswählen zwischen einem durch eine Operation 571 erzeugten Wert und einem vorgegebenen Wert (z. B. Null) enthält. Wie oben, können die Multiplexer 565–567 sich unter einer gemeinsamen Steuerung befinden oder unabhängig gesteuert werden.

Die Durchleitlogik gemäß Fig. 5C (z. B. Multiplexer 575–576) wählt zwischen einem Datenelement eines der

Operanden und einem Identitätsfunktionswert 590 aus. Der Identitätsfunktionswert 590 wird grundsätzlich so ausgewählt, daß das Ergebnis der Durchführung der Operation 572 zwischen dem Identitätsfunktionswert 590 und dem Datenelement identisch dem Wert des Datenelements ist. Wenn beispielsweise die Operation 572 eine Multiplikationsoperation wäre, dann wäre der Identitätsfunktionswert 590 eine Eins. In ähnlicher Weise wäre dann, wenn die Operation 572 eine Additionsoperation wäre, der Identitätsfunktionswert 590 eine Null. Auf diese Weise kann der Wert eines Datenelements selektiv zu dem logischen Zielregister 430 hindurchgeleitet werden, indem die entsprechenden Multiplexer 575–577 veranlaßt werden, den Identitätsfunktionswert 590 auszugeben.

Bei den oben beschriebenen Ausführungsbeispielen war die Schaltung fest verdrahtet, so daß die Operationen partieller Breite an dem am geringsten bewerteten Datenelementabschnitt durchgeführt wurden. Es ist klar, daß die Operation an anderen Datenelementabschnitten als die dargestellten durchgeführt werden kann. Darüber hinaus können, wie oben beschrieben wurde, die zu bearbeitenden Datenelemente durch Software konfigurierbar gemacht werden, indem sämtliche der Operationen mit einem Multiplexer oder dergleichen und nicht nur eine Untermenge der Operationen, wie es in den Fig. 5A–5C dargestellt ist, gekoppelt werden. Während eine Durchleit- und eine Löschlogik als zwei Optionen zum Behandeln der sich ergebenden Datenelemente, die Operationen entsprechen, die nicht berücksichtigt werden sollen, beschrieben werden, können darüber hinaus alternative Ausführungsbeispiele andere Techniken benutzen. Beispielsweise kann ein QNaN als einer der Operanden einer Operation eingegeben werden, deren Ergebnis vernachlässigt werden soll. Auf diese Weise leiten Operationen, die mit dem IEEE 754-Standard, IEEE std. 754-1985, veröffentlicht am 21. März 1985, übereinstimmen, ein NaN zu dem Ergebnis weiter, ohne eine arithmetische Ausnahme auszulösen.

Während keine deutliche Geschwindigkeitserhöhung bei den oben beschriebenen Ausführungsbeispielen erreicht wird, da die volle Breite der Operanden parallel verarbeitet werden kann, ist es klar, daß der Stromverbrauch reduziert werden kann, indem diejenigen der Operationen abgeschaltet werden, deren Ergebnisse vernachlässigt werden. So können beträchtliche Energieeinsparungen erreicht werden. Darüber hinaus kann bei Verwendung von QNaNs und/oder von Identitätsfunktionswerten ein vorhersagbares Ausnahmемодell aufrechterhalten werden, indem vermieden wird, daß Ausnahmen durch Datenelemente ausgelöst werden, die nicht Teil der Operation an gepackten Daten partieller Breite sind. Folglich sind die berichteten Ausnahmen auf diejenigen begrenzt, die in Verbindung mit denjenigen Datenelementen entstehen, an welchen die Operation für gepackte Daten partieller Breite vorgibt zu arbeiten.

Fig. 6 veranschaulicht eine aktuelle Prozessorimplementierung einer arithmetisch-logischen Einheit (ALU), die zum Ausführen von Befehlen für gepackte Daten voller Breite verwendet werden kann. Die ALU gemäß Fig. 6 enthält die Schaltung, die zum Ausführen von Operationen an der vollen Breite der Operanden (d. h. an sämtlichen Datenelementen) erforderlich ist. Fig. 6 zeigt darüber hinaus, daß die ALU mehrere unterschiedliche Arten von Ausführungseinheiten enthalten kann. Bei diesem Beispiel enthält die ALU zwei unterschiedliche Arten von Ausführungseinheiten zum Ausführen unterschiedlicher Arten von Operationen (z. B. verwenden bestimmte ALUs separate Einheiten zum Durchführen von ADD- und MUL-Operationen). Die ADD-Ausführungseinheit und die MUL-Ausführungseinheit sind jeweils in der Lage, als vier separate ADD-Ausfüh-

rungseinheiten bzw. vier separate MUL-Ausführungseinheiten betrieben zu werden. Alternativ kann die ALU eine oder mehrere Multipliziere-Akkumuliere(MAC)-Einheiten enthalten, die jeweils in der Lage sind, mehr als nur eine Art einer Operation durchzuführen. Während die folgenden Beispiele die Verwendung von ADD- und MUL-Ausführungseinheiten und Gleitkommaoperationen unterstellen, ist es klar, daß andere Ausführungseinheiten, wie beispielsweise MAC und/oder Ganzzahloperationen ebenso gut verwendet werden können. Ferner kann es vorzuziehen sein, eine Implementierung partieller Breite zu benutzen (z. B. eine Implementierung mit weniger als einer Eins-zu-eins-Korrespondenz zwischen den Ausführungseinheiten und den Datenelementen) sowie eine zusätzliche Logik, zum Koordinieren der erneuten Verwendung der Ausführungseinheiten, wie es unten beschrieben wird.

#### Hardware partieller Breite und "gestaffelte Ausführung"

Die Fig. 7A-7B veranschaulichen eine Operation für gepackte Daten voller Breite bzw. eine in einer "gestaffelten" Weise durchgeführte Operation für gepackte Daten partieller Breite. Im Kontext dieses Ausführungsbeispiels bezieht sich die "gestaffelte Ausführung" auf den Prozeß des Unterteilens jedes der Operanden eines Befehls in separate Segmente und die sequentielle Verarbeitung jedes Segments unter Verwendung derselben Hardware. Die Segmente werden sequentiell verarbeitet, indem eine Verzögerung in die Verarbeitung aufeinanderfolgender Segmente eingeführt wird. Wie es in den Fig. 7A-7B veranschaulicht ist, werden in beiden Fällen die gepackten Datenoperanden in ein "hoch bewertetes Segment" (Datenelemente 3 und 2) und ein "niedrig bewertetes Segment" (Datenelemente 1 und 0) unterteilt. Bei dem Beispiel gemäß Fig. 7A wird das niedrig bewertete Segment verarbeitet, während das hoch bewertete Segment verzögert wird. Nachfolgend wird das hoch bewertete Segment verarbeitet, und es wird ein Ergebnis voller Breite erzielt. Bei dem Beispiel gemäß Fig. 7B wird das niedrig bewertete Segment verarbeitet, während die Tatsache, ob das hoch bewertete Datensegment verarbeitet wird, von der Implementierung abhängig ist. Beispielsweise braucht das hoch bewertete Datensegment nicht verarbeitet zu werden, sofern das zugehörige Ergebnis auf Null gesetzt werden soll. Zusätzlich ist es klar, daß dann, wenn das hoch bewertete Datensegment nicht verarbeitet wird, sowohl das hoch als auch das niedrig bewertete Datensegment zur gleichen Zeit bearbeitet werden können. In ähnlicher Weise können bei einer Implementierung voller Breite (z. B. einer Implementierung mit einer Eins-zu-Eins-Korrespondenz zwischen Ausführungseinheiten und Datenelementen) die hoch und niedrig bewerteten Datensegmente gleichzeitig verarbeitet werden, wie es in Fig. 7A gezeigt ist.

Obwohl im folgenden nur Ausführungsbeispiele beschrieben werden, die nur ADD- und MUL-Ausführungseinheiten aufweisen, können darüber hinaus auch andere Arten von Ausführungseinheiten, wie beispielsweise MAC-Einheiten, verwendet werden.

Während es eine Reihe unterschiedlicher Wege gibt, auf welchen die gestaffelte Ausführung von Befehlen erreicht werden kann, beschreiben die folgenden Abschnitte zwei exemplarische Ausführungsbeispiele, um diesen Aspekt der Erfindung zu veranschaulichen. Insbesondere empfangen beide der beschriebenen Ausführungsbeispiele jeweils denselben Makrobefehl, der 128-Bit-Operanden enthaltende logische Register spezifiziert.

Bei dem ersten Ausführungsbeispiel bewirkt jeder Makrobefehl, der 128-Bit-Operanden enthaltende logische Register spezifiziert, daß auf die volle Breite der Operanden

aus den physikalischen Registern zugegriffen wird. Nach dem Zugriff auf die Operanden voller Breite aus den Registern werden die Operanden in die niedrig und hoch bewertete Segmente unterteilt (z. B. unter Verwendung von Latch-Speichern und Multiplexern) und sequentiell unter Verwendung derselben Hardware ausgeführt. Die resultierenden Ergebnisse halber Breite werden gesammelt und dann gleichzeitig in ein einziges logisches Register geschrieben.

Im Unterschied dazu wird bei dem zweiten Ausführungsbeispiel jeder 128-Bit-Operanden enthaltende logische Register spezifizierende Makrobefehl in zumindest zwei Mikrobefehle unterteilt, die jeweils an nur der Hälfte der Operanden operieren. Somit werden die Operanden in ein hoch und ein niedrig bewertetes Segment unterteilt und jeder Mikrobefehl bewirkt separat, daß nur auf die Hälfte der Operanden aus den Registern zugegriffen wird. Diese Art einer Unterteilung ist bei einer SIMD-Architektur möglich, weil jeder der Operanden unabhängig von dem anderen ist. Während Implementierungen des zweiten Ausführungsbeispiels die Mikrobefehle in einer beliebigen Reihenfolge (entweder in einem Inner-Reihenfolge oder einem Außer-der-Reihe-Ausführungsmodell) ausführen können, bewirken die jeweiligen Mikrobefehle, daß die von dem Makrobefehl spezifizierte Operation unabhängig oder separat an dem niedrig und hoch bewerteten Segment der Operanden durchgeführt wird. Darüber hinaus bewirkt jeder Mikrobefehl, daß die Hälfte der sich ergebenden Operanden in ein einziges logisches Zielregister, das durch den Makrobefehl spezifiziert ist, geschrieben wird.

Während Ausführungsbeispiele beschrieben werden, bei welchen 128-Bit-Operanden in zwei Segmente unterteilt werden, könnten alternative Ausführungsbeispiele größere oder kleinere Operanden verwenden und/oder diese Operanden in mehr als zwei Segmente unterteilen. Zusätzlich könnten, obwohl zwei Ausführungsbeispiel zum Durchführen einer gestaffelten Ausführung beschrieben wurden, alternative Ausführungsbeispiele andere Techniken verwenden.

#### Erstes eine "gestaffelte Ausführung" benutzendes Ausführungsbeispiel

Fig. 8A veranschaulicht eine Schaltungsanordnung innerhalb eines Prozessors gemäß einem ersten Ausführungsbeispiel, die auf Operanden voller Breite aus den logischen Registern zugreift, aber die die Operationen jeweils an der Hälfte der Breite der Operanden durchführt. Dieses Ausführungsbeispiel unterstellt, daß die Prozessorausführungsmaschine in der Lage ist, einen Befehl pro Taktzyklus zu verarbeiten. Als Beispiel sei angenommen, daß die folgende Sequenz von Befehlen ausgeführt wird: ADD X,Y; MUL A,B. Zum Zeitpunkt T werden jeweils 128 Bits von X und 128 Bits von Y aus den jeweiligen physikalischen Registern über die Ports 1 und 2 gewonnen bzw. gelesen. Die niedriger bewerteten Datensegmente, nämlich die unteren 64 Bits, von sowohl X als auch Y werden an die Multiplexer 802 und 804 weitergeleitet und dann weiter zu den Ausführungseinheiten für die Verarbeitung. Die höher bewerteten Datensegmente, nämlich die höheren 64 Bits von X und Y werden in den Verzögerungselementen M1 und M2 gehalten. Zum Zeitpunkt T+1 werden die höher bewerteten Datensegmente von X und Y aus den Verzögerungselementen M1 und M2 gelesen und an die Multiplexer 802 und 804 weitergeleitet und dann weiter zu den Ausführungseinheiten zur Verarbeitung. Grundsätzlich gestattet der Verzögerungsmechanismus des Speicherns der höher bewerteten Datensegmente in den Verzögerungselementen M1 und M2 einer N-Bit-Hardware (N=64 in diesem Beispiel), 2N-Bits von Daten zu verarbeiten. Die niedrig bewerteten Ergebnisse aus der Ausfüh-

rungseinheit werden in dem Verzögerungselement M3 gehalten, bis die höher bewerteten Ergebnisse bereit sind. Die Ergebnisse beider Verarbeitungsschritte werden dann in die Registerdatei 800 über den Port 3 zurückgeschrieben. Man erinnere sich, daß im Falle einer Operation für gepackte Daten partieller Breite ein oder mehrere Datenelemente des niedrig oder hoch bewerteten Ergebnisses auf einen vorgegebenen Wert (z. B. Null oder den Wert eines entsprechenden Datenelements in entweder X oder Y) anstelle der Ausgabe der ADD- oder MUL-Operation gezwungen werden können.

Zum Zeitpunkt T+1 kann darüber hinaus der MUL-Befehl gestartet worden sein. Somit können zum Zeitpunkt T+1 128 Bits von A und B jeweils von ihren zugehörigen Registern über die Ports 1 und 2 gelesen worden sein. Die niedriger bewerteten Datensegmente, nämlich die unteren 64 Bits, von sowohl A als auch B, können in die Multiplexer 806 und 808 geleitet werden. Nachdem die höher bewerteten Bits von X und Y aus den Verzögerungselementen M1 und M2 entfernt und in die Multiplexer 806 und 808 geleitet worden sind, können die höher bewerteten Bits von A und B im Speicher der Verzögerungselemente M1 und M2 gehalten werden. Die Ergebnisse beider Verarbeitungsschritte werden dann in die Registerdatei 800 über den Port 3 zurückgeschrieben.

Somit werden gemäß einem Ausführungsbeispiel der Erfindung Ausführungseinheiten vorgesehen, die nur die Hälfte der Hardware enthalten (z. B. zwei ADD-Ausführungseinheiten einfacher Genauigkeit und zwei MUL-Ausführungseinheiten einfacher Genauigkeit) anstelle der Ausführungseinheiten, die zur parallelen Verarbeitung der vollen Breite der Operanden erforderlich sind, wie sie in einem gegenwärtigen Prozessor zu finden sind. Diese Ausführungsform zieht Vorteil aus einer statistischen Analyse, die zeigt, daß Multimedia-Anwendungen etwa 50% ADD-Befehle und 50% MUL-Befehle verwenden. Auf der Grundlage dieser Statistik unterstellt dieses Ausführungsbeispiel, daß Multimedia-Befehle grundsätzlich dem folgenden Muster folgen: ADD, MUL, ADD, MUL, etc. Durch Verwenden der ADD und MUL-Ausführungseinheiten in der oben beschriebenen Weise sorgt die vorliegende Erfindung für eine optimale Verwendung der Ausführungseinheiten, wobei sie eine mit gegenwärtigen Prozessoren vergleichbare Leistung zu niedrigeren Kosten ermöglicht.

Fig. 8B ist ein Zeitdiagramm, das die Schaltung gemäß Fig. 8A näher veranschaulicht. Wie es in Fig. 8B veranschaulicht ist, führen die beiden ADD-Ausführungseinheiten dann, wenn der Befehl "ADD X,Y" zum Zeitpunkt T ausgegeben wird, zunächst eine Addition der niedriger bewerteten Datensegmente oder der niedriger bewerteten beiden gepackten Datenelement gemäß Fig. 1, nämlich  $X_0Y_0$  und  $X_1Y_1$  aus. Zum Zeitpunkt T+1 wird die ADD-Operation an den verbleibenden zwei Datenelementen der Operanden durch dieselben Ausführungseinheiten durchgeführt, und die nachfolgenden zwei Datenelemente des höher bewerteten Datensegments werden addiert, nämlich  $X_2Y_2$  und  $X_3Y_3$ . Während das obige Ausführungsbeispiel unter Bezugnahme auf ADD- und MUL-Operationen unter Verwendung von zwei Ausführungseinheiten beschrieben wurde, können alternative Ausführungsbeispiele eine beliebige Anzahl von Ausführungseinheiten verwenden und/oder eine beliebige Anzahl unterschiedlicher Operationen in einer gestaffelten Weise ausführen.

Gemäß diesem Ausführungsbeispiel kann eine 64-Bit-Hardware verwendet werden, um 128-Bit-Daten zu verarbeiten. Ein 128-Bit-Register kann in vier 32-Bit-Elemente unterteilt werden, von denen jedes einen separaten 32-Bit-Wert repräsentiert. Zum Zeitpunkt T führen die beiden

ADD-Ausführungseinheiten zunächst Additionen an den zwei niedrigen 32-Bit-Werten durch, gefolgt von einer Addition an den höheren 32-Bit-Werten zum Zeitpunkt T+1. Im Falle einer MUL-Operation verhalten sich die MUL-Ausführungseinheiten in der gleichen Weise. Diese Fähigkeit zu einer Verwendung gegenwärtig verfügbarer 64-Bit-Hardware, um 128-Bit-Daten zu verarbeiten, stellt einen signifikanten Kostenvorteil für Hardware-Hersteller dar.

Wie oben beschrieben, werden die ADD- und MUL-Ausführungseinheiten gemäß dem vorliegenden Ausführungsbeispiel erneut verwendet, um eine zweite ADD- oder MUL-Operation bei einem nachfolgenden Taktzyklus erneut auszuführen. Selbstverständlich werden im Fall eines Befehls für gepackte Daten partieller Breite die Ausführungseinheiten erneut verwendet, aber die Operation wird nicht notwendigerweise erneut ausgeführt, da die Stromversorgung zur Ausführungseinheit selektiv abgeschaltet werden kann. In jedem Fall zieht, wie oben beschrieben wurde, dieses Ausführungsbeispiel für diese Neu-Verwendung bzw. "gestaffelte Ausführung" Vorteil aus dem statistischen Verhalten von Multimedia-Anwendungen.

Wenn ein zweiter ADD-Befehl einem ersten ADD-Befehl folgt, kann der zweite ADD-Befehl durch eine Einplanungseinheit (scheduling unit) verzögert werden, um den ADD-Ausführungseinheiten zu gestatten, den ersten ADD-Befehl abzuschließen, bzw. genauer, den ADD-Befehl für das höher bewertete Datensegment abzuschließen. Der zweite ADD-Befehl kann dann mit der Ausführung beginnen. Alternativ kann bei einem Außer-der-Reihe-Prozessor die Einplanungseinheit bestimmen, daß ein MUL-Befehl, der sich weiter hinten im Befehlsstrom befindet, außer der Reihe ausgeführt werden kann. Wenn dies der Fall ist, kann die Einplanungseinheit die MUL-Ausführungseinheiten informieren, mit der Verarbeitung des MUL-Befehls zu beginnen. Wenn keine MUL-Befehle zur Verarbeitung zum Zeitpunkt T+1 verfügbar sind, gibt der Einplaner keinen dem ersten ADD-Befehl folgenden Befehl aus, womit den ADD-Ausführungseinheiten Zeit gegeben wird, den ersten ADD-Befehl abzuschließen, bevor der zweite ADD-Befehl begonnen wird.

Noch ein weiteres Ausführungsbeispiel der Erfindung gestattet, daß "back-to-back"-ADD- oder -MUL-Befehle ausgegeben werden, indem die Befehle auf den gleichen Ausführungseinheiten in halben Taktzyklen anstelle von vollen Taktzyklen ausgeführt werden. Die Ausführung eines Befehls in einem halben Taktzyklus "verdoppelt" effektiv die Hardware, d. h. macht die Hardware zweimal so schnell. Auf diese Weise können die ADD- oder MUL-Ausführungseinheiten während jedes Taktzyklus verfügbar sein, um einen neuen Befehl zu verarbeiten. Eine verdoppelte Hardware gestattet, daß die Hardwareeinheiten zweimal so effektiv ausführen, wie eine einfache Hardware, die nur bei vollständigen Taktzyklen ausführt. Eine derart verdoppelte Hardware erfordert signifikant mehr Hardware, kann jedoch effektiv den Befehl im halben Taktzyklus verarbeiten.

Es ist klar, daß Modifikationen und Variationen der Erfindung von der obigen Lehre abgedeckt werden und innerhalb des Umfangs der anhängigen Ansprüche liegen können, ohne vom Geist und Umfang der Erfindung abzuweichen. Beispielsweise können, obwohl nur zwei Ausführungseinheiten oben beschrieben wurden, beliebig viele logische Einheiten zur Verfügung gestellt werden.

Zweites eine "gestaffelte Ausführung" verwendendes Ausführungsbeispiel

Gemäß einem alternativen Ausführungsbeispiel der Erfindung wird die gestaffelte Ausführung eines Operanden

voller Breite erreicht, indem ein Makrobefehl für eine volle Breite in zumindest zwei Mikrobefehle konvertiert wird, die jeweils an nur der Hälfte der Operanden operieren. Wie unten näher beschrieben wird, kann dann, wenn die Makrobefehle eine Operation für gepackte Daten partieller Breite spezifizieren, eine bessere Leistung erreicht werden, indem Mikrobefehle eliminiert werden, die zum Bestimmen des Ergebnisses partieller Breite nicht erforderlich sind. Auf diese Weise werden die Einschränkungen der Prozessorressourcen reduziert, und der Prozessor wird nicht unnötigerweise mit wirkungslosen Mikrobefehlen okkupiert. Obwohl die folgende Beschreibung unter Berücksichtigung eines speziellen Registerumbenennungsverfahrens beschrieben ist, ist es klar, daß andere Registerumbenennungsmechanismen ebenfalls im Rahmen der Erfindung benutzt werden können. Das unten beschriebene Registerumbenennungsverfahren unterstellt die Verwendung einer Register-Alias-Tabelle (RAT), eines Umordnungspuffers (ROB) und eines Retirement-Puffers, wie sie im Detail in dem US-Patent Nr. 5,446,912 beschrieben sind. Alternative Registerumbenennungsverfahren, wie sie beispielsweise im US-Patent Nr. 5,197,132 beschrieben sind, können ebenfalls implementiert werden.

Fig. 9 veranschaulicht ein Ausführungsbeispiel einer Pipeline zum Durchführen von Operationen an Operanden in einer "gestaffelten" Weise, indem ein Makrobefehl in eine Mehrzahl von Mikrobefehlen konvertiert wird, die jeweils einen Abschnitt der Operanden voller Breite bearbeiten. Es sei angemerkt, daß verschiedene andere Stufen der Pipeline, zum Beispiel eine "Vorab-Heranzahl-Stufe", nicht im Detail gezeigt sind, um die Erfindung nicht unnötigerweise zu verdunkeln. Wie veranschaulicht, wird an der Dekodierstufe der Pipeline ein Makrobefehl für eine volle Breite empfangen, der logische Quellregister spezifiziert, die jeweils einen Operanden voller Breite (z. B. 128 Bits) speichern. Beispielsweise sind die beschriebenen Operanden gepackte 128-Bit-Gleitkomma-Datenoperanden. Bei diesem Beispiel unterstützt der Prozessor Y logische Register zum Speichern gepackter Gleitkomma-Daten. Der Makrobefehl wird in Mikrobefehle konvertiert, nämlich eine "hoch bewertete Operation" und eine "niedrig bewertete Operation", die jeweils bewirken, daß die Operation des Makrobefehls an der halben Breite der Operanden (z. B. 64 Bits) durchgeführt wird.

Die zwei Mikrobefehle für die halbe Breite bewegen sich dann in eine Registerumbenennungsstufe der Pipeline. Die Registerumbenennungsstufe enthält eine Anzahl von Registerabbildungs- und Umordnungspuffern. Die logischen Quellregister jedes Mikrobefehls sind Zeiger auf spezielle Registereinträge in einer Registerabbildungstabelle (z. B. einer RAT). Die Einträge in der Registerabbildungstabelle wiederum zeigen auf den Ort des physikalischen Quellspeicherplatzes in einem ROB oder in einem Retirement-Register. Gemäß einem Ausführungsbeispiel wird, um die oben beschriebenen hoch und niedrig bewerteten Operationen halber Breite unterzubringen, eine RAT für gepackte Gleitkomma-Daten mit  $Y \cdot 2$  Einträgen vorgesehen. Somit wird beispielsweise anstelle einer RAT mit den Einträgen für 8 logische Register eine RAT mit 16 Einträgen erzeugt, wobei jeder Eintrag entweder als "hoch" oder als "niedrig" adressiert wird. Jeder Eintrag identifiziert eine 64-Bit-Quelle, die entweder einem hohen oder einem niedrigen Teil des logischen 128-Bit-Registers entspricht.

Jeder der hoch und niedrig bewerteten Mikrobefehle weist somit zugehörige Einträge in der Registerabbildungstabelle auf, die den jeweiligen Operanden entsprechen. Die Mikrobefehle werden dann in eine Einplanungsstufe (bei einem Außer-der-Reihe-Prozessor) oder an eine Ausführungsstufe (bei einem In-der-Reihenfolge-Prozessor) bewegt. Je-

der Mikrobefehl liest und verarbeitet separat ein 64-Bit-Segment der 128-Bit-Operanden. Eine der Operationen (z. B. die niedriger bewertete Operation) wird zunächst von den 64-Bit-Hardware-Einheiten ausgeführt. Dann führt dieselbe 64-Bit-Hardware-Einheit die höher bewertete Operation aus. Es ist klar, daß kein Befehl oder auch mehrere Befehle zwischen der niedriger und der höher bewerteten Operation ausgeführt werden können.

Obwohl das obige Ausführungsbeispiel beschreibt, daß der Makrobefehl in zwei Mikrobefehle unterteilt wird, können alternative Ausführungsbeispiele den Makrobefehl in mehr als zwei Mikrobefehle unterteilen. Während Fig. 9 zeigt, daß die gepackten Gleitkomma-Daten an eine Retirement-Registerdatei mit  $Y \cdot 2$  64-Bit-Registern, die jeweils als hoch oder niedrig bezeichnet sind, zurückgibt, können alternative Ausführungsbeispiele eine Retirement-Registerdatei mit  $Y$  128-Bit-Registern verwenden. Zusätzlich können, während ein Ausführungsbeispiel beschrieben ist, das einen Registerumbenennungsmechanismus mit einem Umordnungspuffer und einer Retirement-Registerdatei aufweist, alternative Ausführungsbeispiele einen beliebigen Registerumbenennungsmechanismus verwenden. Beispielsweise verwendet der Registerumbenennungsmechanismus gemäß dem US-Patent Nr. 5,197,132 eine Vorgeschichte-Warteschlange und eine Back-up-Abbildung.

Fig. 10 ist ein Zeitdiagramm, das das in Fig. 9 beschriebene Ausführungsbeispiel näher veranschaulicht. Zum Zeitpunkt T tritt ein Makrobefehl "ADD X,Y" in die Dekodierstufe der Pipeline gemäß Fig. 9 ein. Beispielsweise ist der Makrobefehl hier ein 128-Bit-Befehl. Der 128-Bit-Makrobefehl wird in zwei 64-Bit-Mikrobefehle konvertiert, nämlich die hoch bewertete Operation "ADD  $X_H, Y_H$ " und die niedrig bewertete Operation "ADD  $X_L, Y_L$ ". Jeder Mikrobefehl bearbeitet dann ein Datensegment, das zwei Datenelemente enthält. Beispielsweise kann zum Zeitpunkt T die niedrig bewertete Operation durch eine 64-Bit-Ausföhrungseinheit ausgeführt werden. Dann wird zu einem anderen Zeitpunkt (z. B. zum Zeitpunkt T+N) die höher bewertete Operation durch dieselbe 64-Bit-Ausföhrungseinheit ausgeführt. Dieses Ausführungsbeispiel der Erfindung ist somit insbesondere für die Verarbeitung von 128-Bit-Befehlen unter Verwendung vorhandener 64-Bit-Hardwaressysteme ohne signifikante Veränderungen der Hardware geeignet. Die vorhandenen Systeme können leicht erweitert werden, so daß sie eine neue Abbildung zum Behandeln gepackter Gleitkomma-Daten zusätzlich zu den vorhandenen logischen Registerabbildungen enthalten.

Es wird jetzt auf Fig. 11 Bezug genommen; anhand der die Dekodierlogik beschrieben wird, die gemäß einem Ausführungsbeispiel der Erfindung verwendet werden kann. Kurz gesagt empfangen bei dem dargestellten Ausführungsbeispiel eine Mehrzahl von Decodierern 1110, 1120 und 1130 jeweils einen Makrobefehl und konvertieren ihn in einen Mikrobefehl. Dann wird die Mikrooperation hinab zum Rest der Pipeline gesendet. Selbstverständlich sind nicht N Mikrobefehle für die Ausführung jedes Makrobefehls erforderlich. Folglich ist es üblicherweise der Fall, daß nur eine Untermenge von Mikrobefehlen zur Verarbeitung durch den Rest der Pipeline eingereicht wird.

Wie oben beschrieben, können gepackte Datenoperationen als zwei Mikrobefehle halber Breite implementiert werden (z. B. eine hoch bewertete Operation und eine niedrig bewertete Operation). Anstelle des unabhängigen Decodierens des Makrobefehls durch zwei Decodierer, um die hoch und die niedrig bewertete Operation zu erzeugen, wie es typischerweise bei bekannten Prozessorimplementierungen erforderlich sein würde, können als Merkmal der vorliegenden Ausführungsform beide Mikrobefehle durch den glei-

chen Decodierer erzeugt werden. Bei diesem Beispiel wird dies durch die Replikationslogik 1150 ausgeführt, welche entweder die hoch bewertete oder die niedrig bewertete Operation repliziert und nachfolgend die sich ergebende replizierte Operation geeignet modifiziert, um die verbleibende Operation zu erzeugen. Wie oben beschrieben wurde, ist es wichtig, daß durch sorgfältiges Codieren des Registeradressesraums die von den Mikrobefehlen adressierten Register (z. B. die logischen Quell- und Zielregister) so gebildet werden können, daß sie sich durch einziges Bit unterscheiden. Im Ergebnis kann die Modifikationslogik 1160 in ihrer einfachsten Form einen oder mehrere Inverter aufweisen, um die richtigen Bits zu invertieren, um eine hoch bewertete Operation aus einer niedrig bewerteten Operation und umgekehrt zu erzeugen. In jedem Fall wird dann der replizierte Mikrobefehl zum Multiplexer 1170 weitergeleitet. Der Multiplexer 1170 empfängt darüber hinaus einen von dem Decodierer 1120 erzeugten Mikrobefehl. Bei diesem Beispiel gibt der Multiplexer unter der Steuerung eines Gültigkeitsdecodierers 1180 den replizierten Mikrobefehl für gepackte Datenoperationen (die Operationen für gepackte Daten partieller Breite umfassen) aus und gibt den Mikrobefehl aus, der vom Decodierer 1120 empfangen wurde, für von gepackten Datenoperationen abweichende Operationen. Folglich ist es vorteilhaft, die Befehlscodeabbildung zu optimieren, um die Erfassung gepackter Datenoperationen durch die Replikationslogik 1150 zu vereinfachen. Wenn beispielsweise nur ein geringer Abschnitt des Makrobefehls überprüft werden muß, um gepackte Datenoperationen von anderen Operationen zu unterscheiden, so könnte eine geringere Schaltungsgröße durch den Gültigkeitsdecodierer 1180 benutzt werden.

Bei einer Implementierung, die Quelldatenelemente zu dem logischen Zielregister zum Zwecke der Ausführung von Operationen für gepackte Daten partieller Daten durchleitet, kann zusätzlich zur Auswahllogik, ähnlich der die unter Bezugnahme auf die Fig. 5A und 5C beschrieben worden ist, Logik enthalten sein, um entweder die hoch oder die niedrig bewertete Operation zu eliminieren (zu "vernichten"). Vorzugsweise wird aus Leistungsgründen der unwesentliche Mikrobefehl früh in der Pipeline eliminiert. Diese Eliminierung kann gemäß dem dargestellten Ausführungsbeispiel ausgeführt werden, indem ein Mikrobefehlsauswahlsignal verwendet wird, das von der Mikrobefehlsängenbestimmungsschaltung 1190 ausgegeben wird. Die Mikrobefehlsängenbestimmungsschaltung 1190 überprüft einen Abschnitt des Makrobefehls und erzeugt das Mikrobefehlsauswahlsignal, welches eine spezielle Kombination einer oder mehrerer Mikrobefehle anzeigt, die weiter unten in der Pipeline ausgeführt werden sollen. Im Falle eines skalaren SIMD-Befehls wird nur einer der sich ergebenden hoch und niedrig bewerteten Operationen gestattet, fortzufahren. Beispielsweise kann das Mikrobefehlsauswahlsignal durch eine Bitmaske repräsentiert sein, die diejenigen der Mikrobefehle identifiziert, die beibehalten werden sollen, und diejenigen, die eliminiert werden sollen. Alternativ kann das Mikrobefehlsauswahlsignal einfach die Nummer der Mikrobefehle von einem vorgegebenen Standpunkt aus anzeigen, die eliminiert bzw. zurückgehalten werden sollen. Die zum Ausführen der oben beschriebenen Beseitigung erforderliche Logik variiert in Abhängigkeit vom Steuerungsmechanismus, der die Mikrobefehle durch den Rest der Pipeline leitet. Wenn beispielsweise die Mikrobefehle in eine Warteschlange eingereiht werden, würde Logik hinzugefügt werden, um den Kopf- und den Ende-Zeiger der Mikrobefehlswarteschlange zu bearbeiten, um zu bewirken, daß ungültige Mikrobefehle durch nachfolgend erzeugte gültige Mikrobefehle überschrieben werden. Zahlreiche weitere Eliminie-

rungstechniken sind Fachleuten auf diesem Gebiet klar.

Obwohl aus Gründen der Vereinfachung nur ein einzelner Makrobefehl gezeigt ist, der zu einem vorgegebenen Zeitpunkt bei dem dargestellten Ausführungsbeispiel decodiert wird, können bei alternativen Ausführungsbeispielen mehrere Makrobefehle gleichzeitig decodiert werden. Darüber hinaus ist es klar, daß die Mikrobefehlsreplikation eine breitere Anwendung findet, als die bei dem obigen Ausführungsbeispiel dargestellte. Beispielsweise könne auf eine Weise, die ähnlich der oben beschriebenen ist, Makrobefehle für gepackte Daten voller Breite und partieller Breite durch denselben Decodierer decodiert werden. Sofern ein Präfix verwendet wird, um Makrobefehle für gepackte Daten voller Breite und partieller Breite zu unterscheiden, kann der Decodierer einfach den Präfix ignorieren und beide Arten von Befehlen auf die gleiche Weise decodieren. Dann werden die geeigneten Bits in den sich ergebenden Mikrooperationen modifiziert, um selektiv die Verarbeitung für entweder sämtliche oder eine Untermenge der Datenelemente zu ermöglichen. Auf diese Weise können Mikrobefehle für gepackte Daten voller Breite aus Mikrobefehlen für gepackte Daten partieller Breite erzeugt werden oder umgekehrt, wodurch sich die Komplexität des Decodierers verringert.

#### Patentansprüche

1. Prozessor mit einer Mehrzahl von Registern und einer Registerumbenennungseinheit, die mit der Mehrzahl von Registern gekoppelt ist, um eine Architekturregisterdatei zum Speichern gepackter Datenoperanden zur Verfügung zu stellen, wobei jeder der gepackten Datenoperanden eine Mehrzahl von Datenelementen aufweist,

**dadurch gekennzeichnet,**

daß ein Decodierer mit der Registerumbenennungseinheit gekoppelt ist, um einen ersten und einen zweiten Satz von Befehlen zu decodieren, die jeweils wenigstens ein Register in der Architekturregisterdatei spezifizieren, wobei jeder Befehl in dem ersten Satz von Befehlen Operationen an sämtlichen in dem wenigstens einen spezifizierten Register gespeicherten Datenelementen spezifiziert, wobei jeder Befehl des zweiten Satzes von Befehlen eine Operation an nur einer Untermenge der in dem wenigstens einen spezifizierten Register gespeicherten Datenelementen spezifiziert, und daß eine Ausführungseinheit partieller Breite mit dem Decodierer gekoppelt ist, um entweder von dem ersten oder dem zweiten Satz von Befehlen spezifizierte Operationen auszuführen.

2. Prozessor nach Anspruch 1, dadurch gekennzeichnet, daß die Untermenge der in einem spezifizierten wenigstens einen Register gespeicherten Datenelemente einander entsprechende am geringsten bewertete Datenelemente umfaßt.

3. Prozessor nach Anspruch 1 oder 2, gekennzeichnet durch eine Ausführungseinheit zum selektiven Ausführen einer spezifizierten Operation an einem oder mehreren Datenelementen in dem spezifizierten wenigstens einen Register in Abhängigkeit davon, ob die spezifizierte Operation dem ersten oder dem zweiten Satz von Befehlen zugeordnet ist.

4. Prozessor nach Anspruch 3, dadurch gekennzeichnet, daß die Ausführungseinheit eine Mehrzahl von Multiplexern aufweist, um zwischen einem Ergebnis der spezifizierten Operation und einem vorgegebenen Wert zu wählen.

5. Der Prozessor nach Anspruch 3, dadurch gekenn-

zeichnet, daß die Ausführungseinheit eine Mehrzahl von Multiplexern aufweist, um zwischen einem der wenigstens einen Datenelemente und einer Identitätsfunktion zur Eingabe an die spezifizierte Operation zu wählen.

6. Verfahren zum Ausführen eines Makrobefehls in einem Prozessor, wobei

ein einzelner Makrobefehl empfangen wird, der zumindest zwei logische Register in einer Registerdatei gepackter Daten spezifiziert, wobei die beiden logischen Register einen ersten gepackten Datenoperanden bzw. einen zweiten gepackten Datenoperanden speichern, die einander entsprechende Datenelemente aufweisen, und

eine erste und eine zweite Mehrzahl der einander entsprechenden Datenelemente aus dem ersten bzw. dem zweiten gepackten Datenoperanden unabhängig zu verschiedenen Zeiten unter Verwendung der gleichen Schaltung verarbeitet werden, um unabhängig eine erste und eine zweite Mehrzahl von Ergebnisdatenelementen zu erzeugen, indem

eine von einem einzelnen Makrobefehl spezifizierte Operation an zumindest einem Paar einander entsprechender Datenelemente in der ersten und der zweiten Mehrzahl entsprechender Datenelemente durchgeführt wird, um zumindest ein Ergebnisdatenelement der ersten und zweiten Mehrzahl von Ergebnisdatenelementen zu erzeugen, und

die verbleibenden Ergebnisdatenelemente der ersten und zweiten Mehrzahl von Ergebnisdatenelementen auf wenigstens einen vorgegebenen Wert gesetzt werden, und

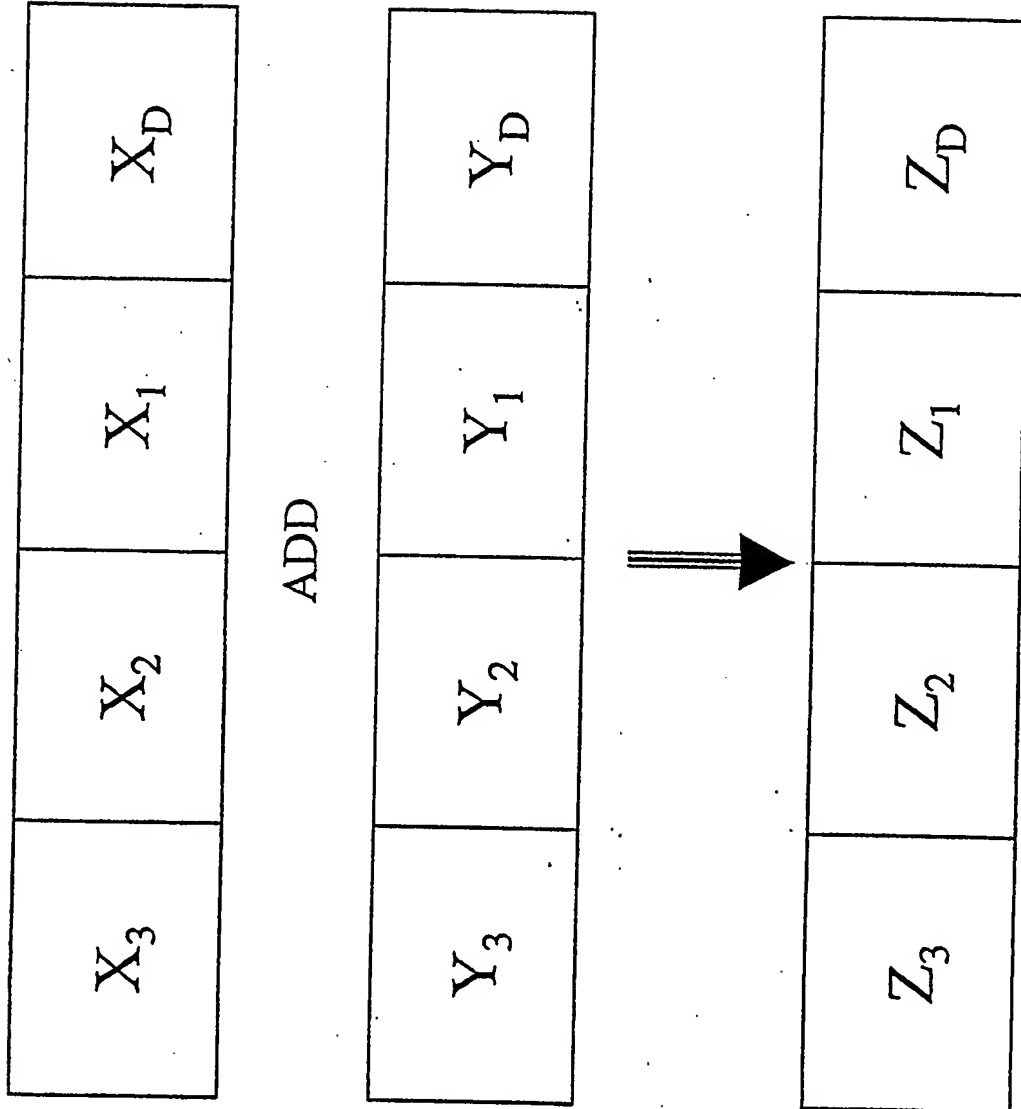
die erste und zweite Mehrzahl von Ergebnisdatenelementen in einem einzigen logischen Register als dritter gepackter Datenoperand gespeichert werden.

7. Verfahren nach Anspruch 6, dadurch gekennzeichnet, daß der wenigstens eine vorgegebene Wert Werte der Datenelemente von entweder dem ersten gepackten Datenoperanden oder dem zweiten gepackten Datenoperanden umfaßt.

8. Verfahren nach Anspruch 6, dadurch gekennzeichnet, daß der wenigstens eine vorgegebene Wert Null ist.

9. Verfahren nach Anspruch 6, dadurch gekennzeichnet, daß der wenigstens eine vorgegebene Wert eine Keine-Zahl(NaN)-Anzeige ist.

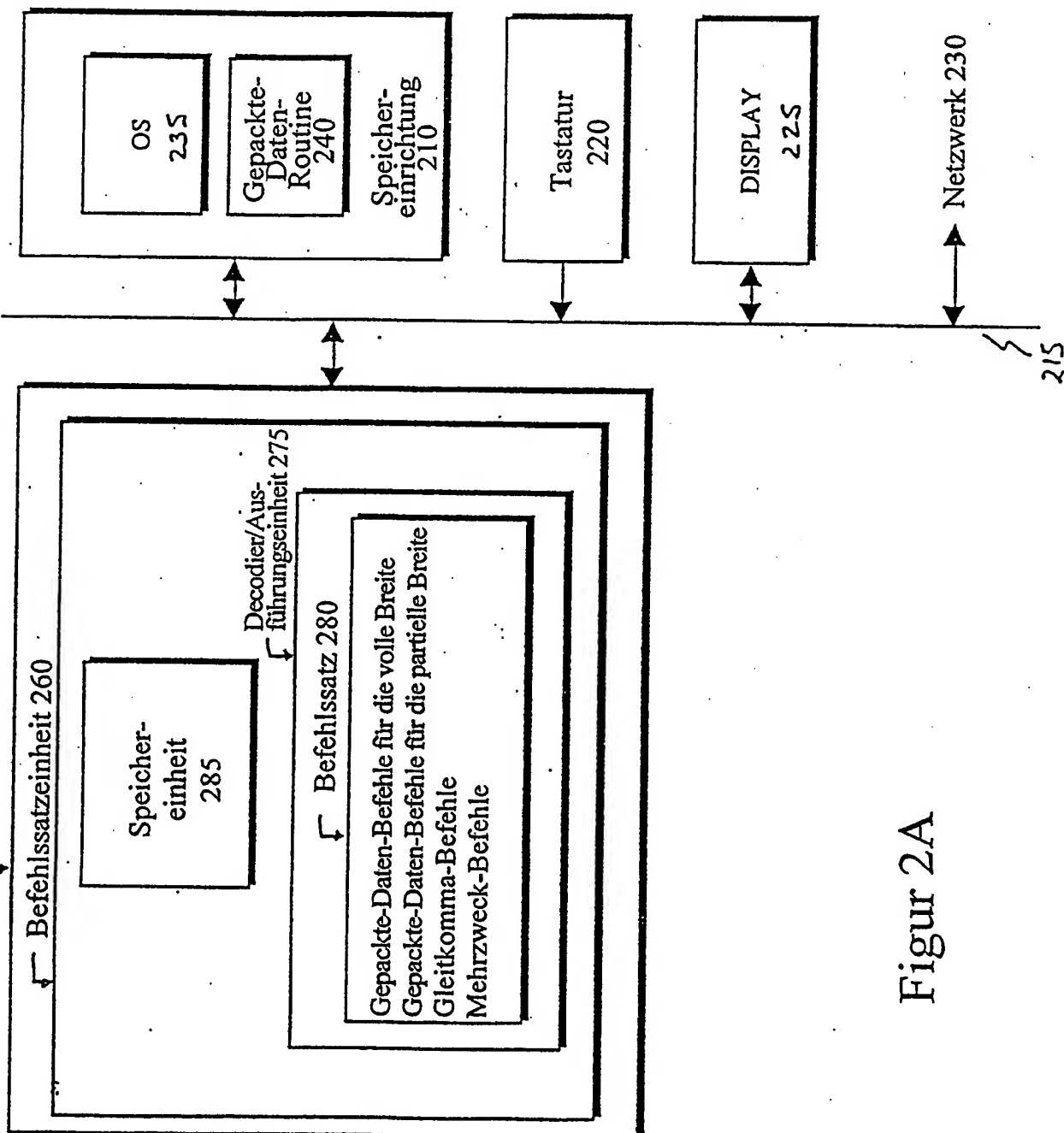
Hierzu 13 Seite(n) Zeichnungen



**FIGUR 1**  
(Stand d. Technik)

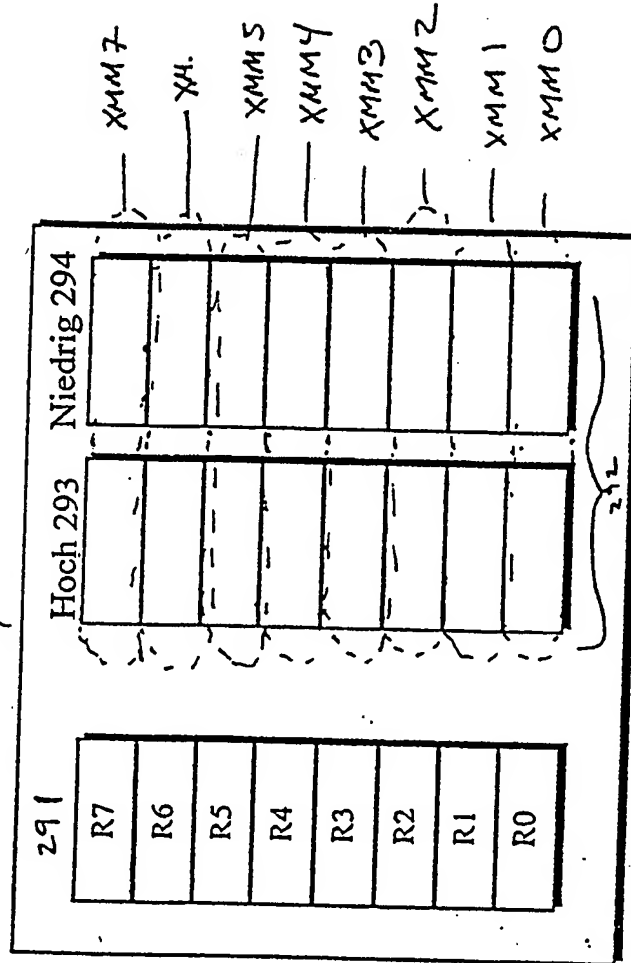
Prozessor 205

Befehlssatz 260



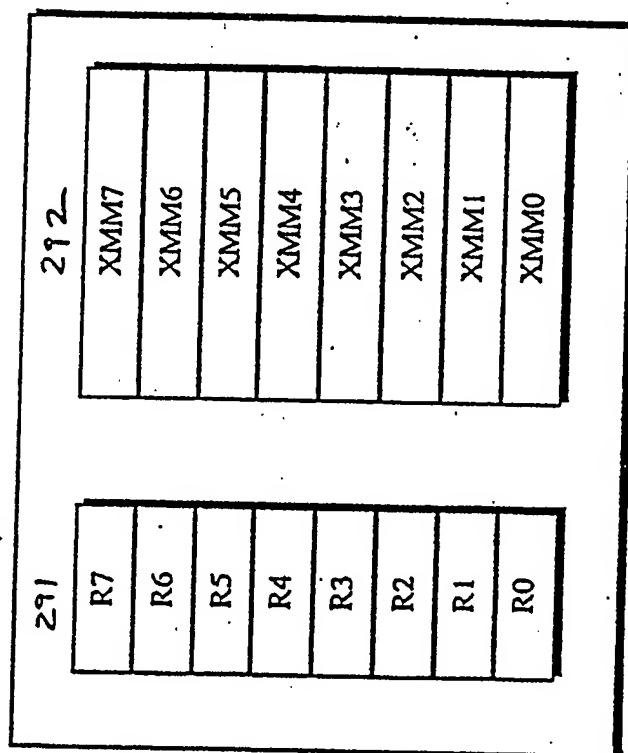
Figur 2A

28S

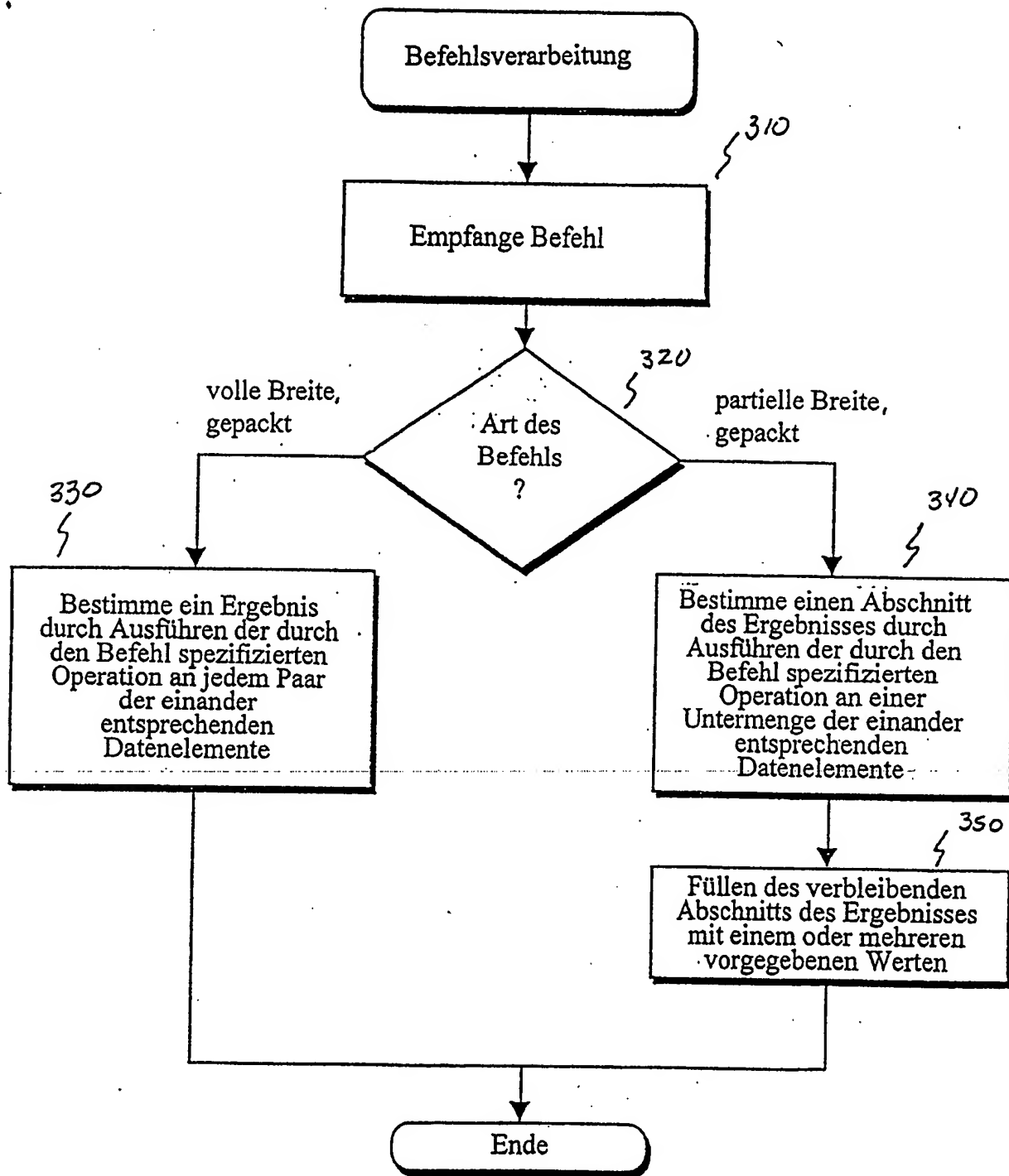


Figur 2C

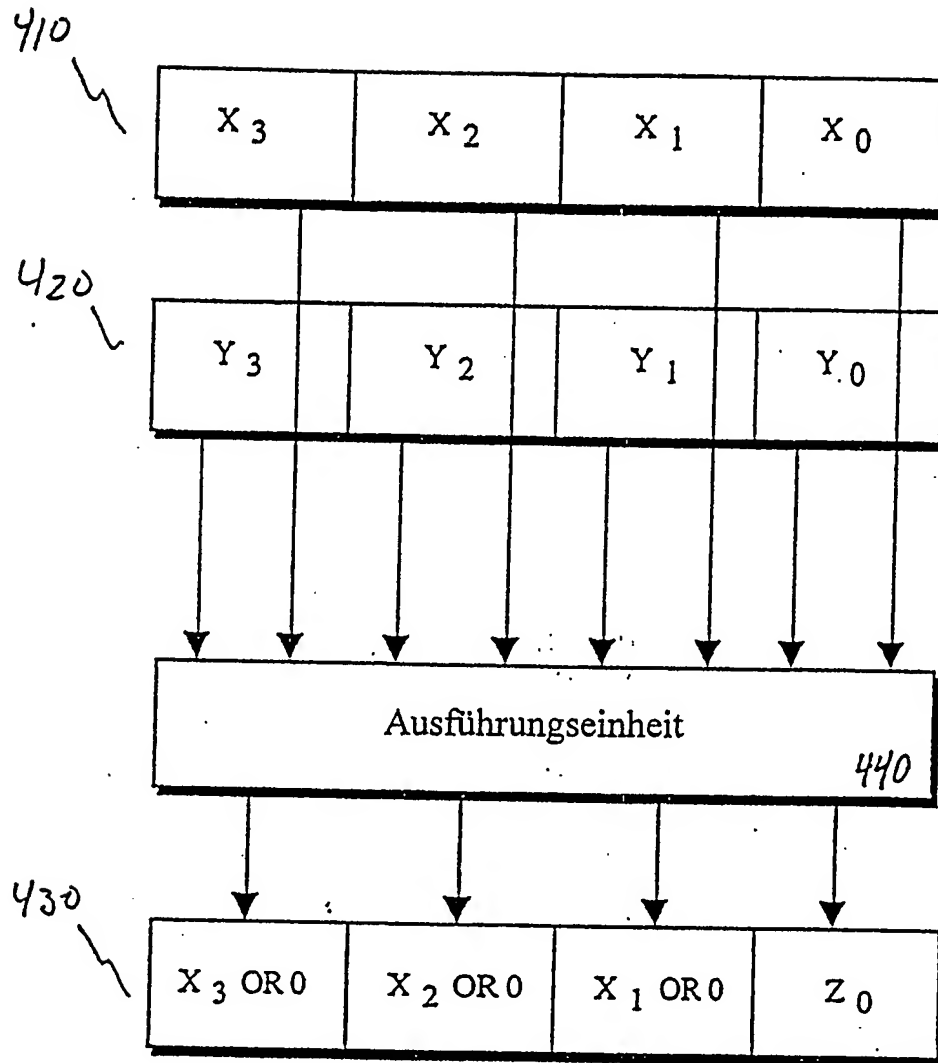
28S



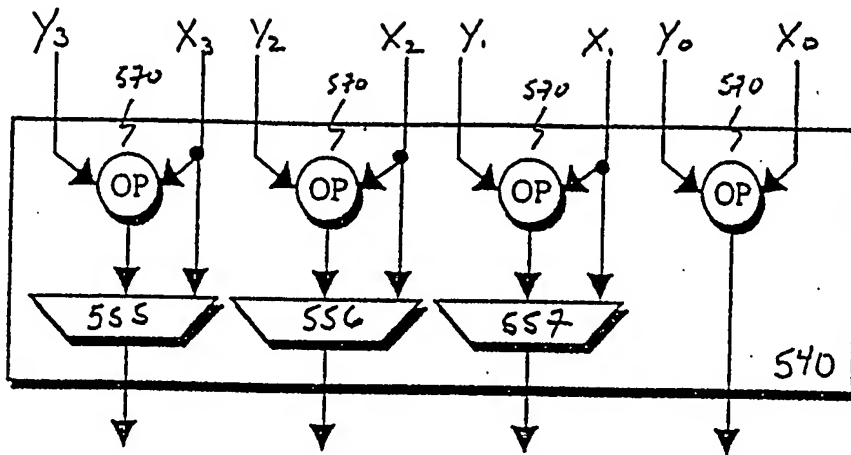
Figur 2B



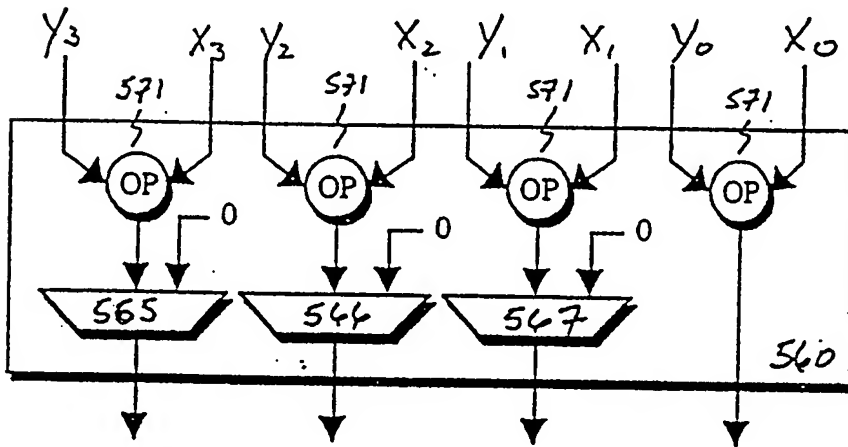
Figur 3



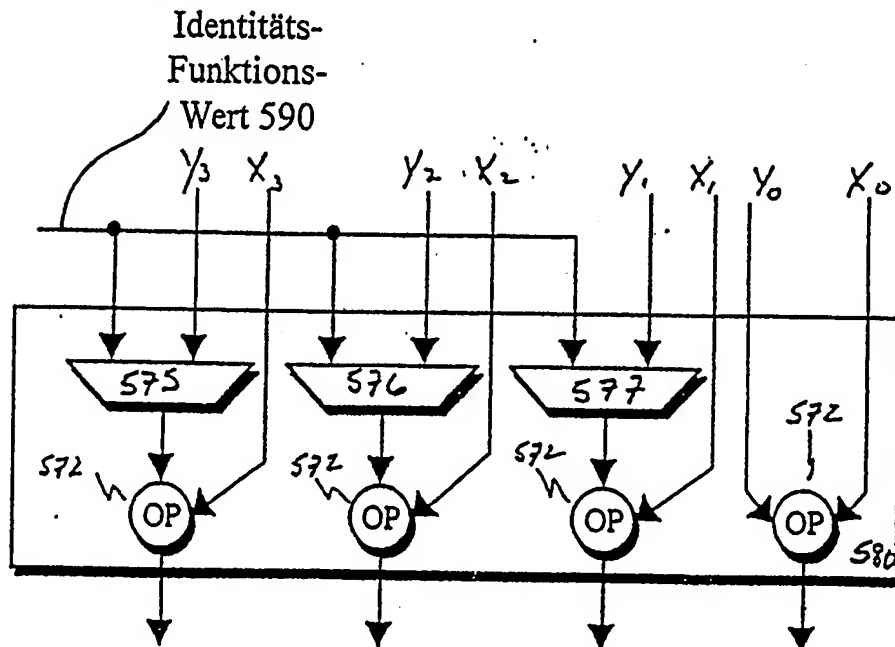
Figur 4



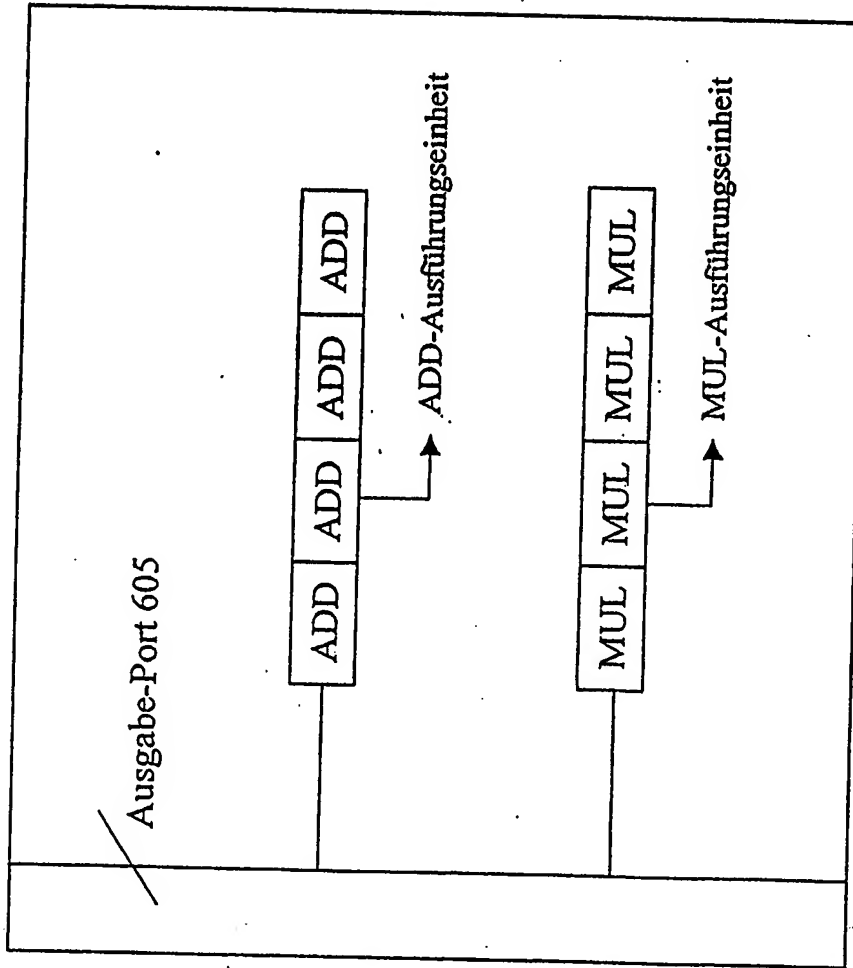
Figur 5A



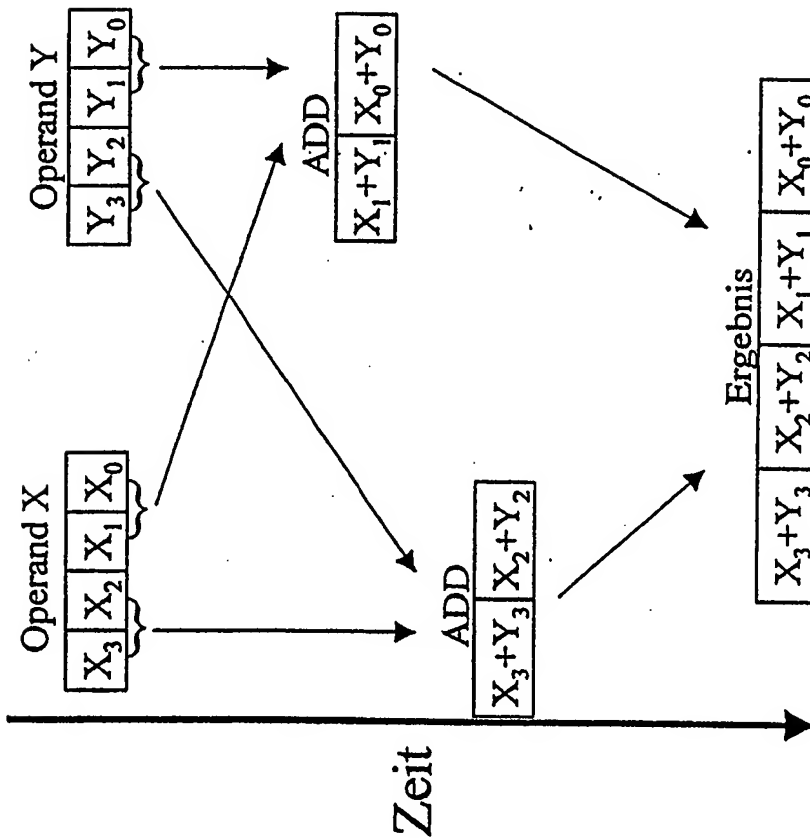
Figur 5B



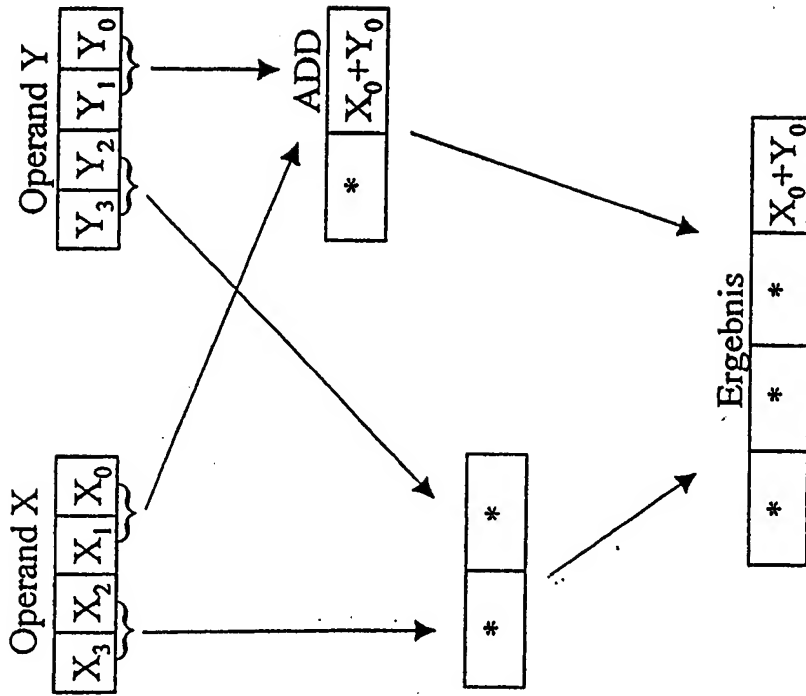
Figur 5C



FIGUR 6



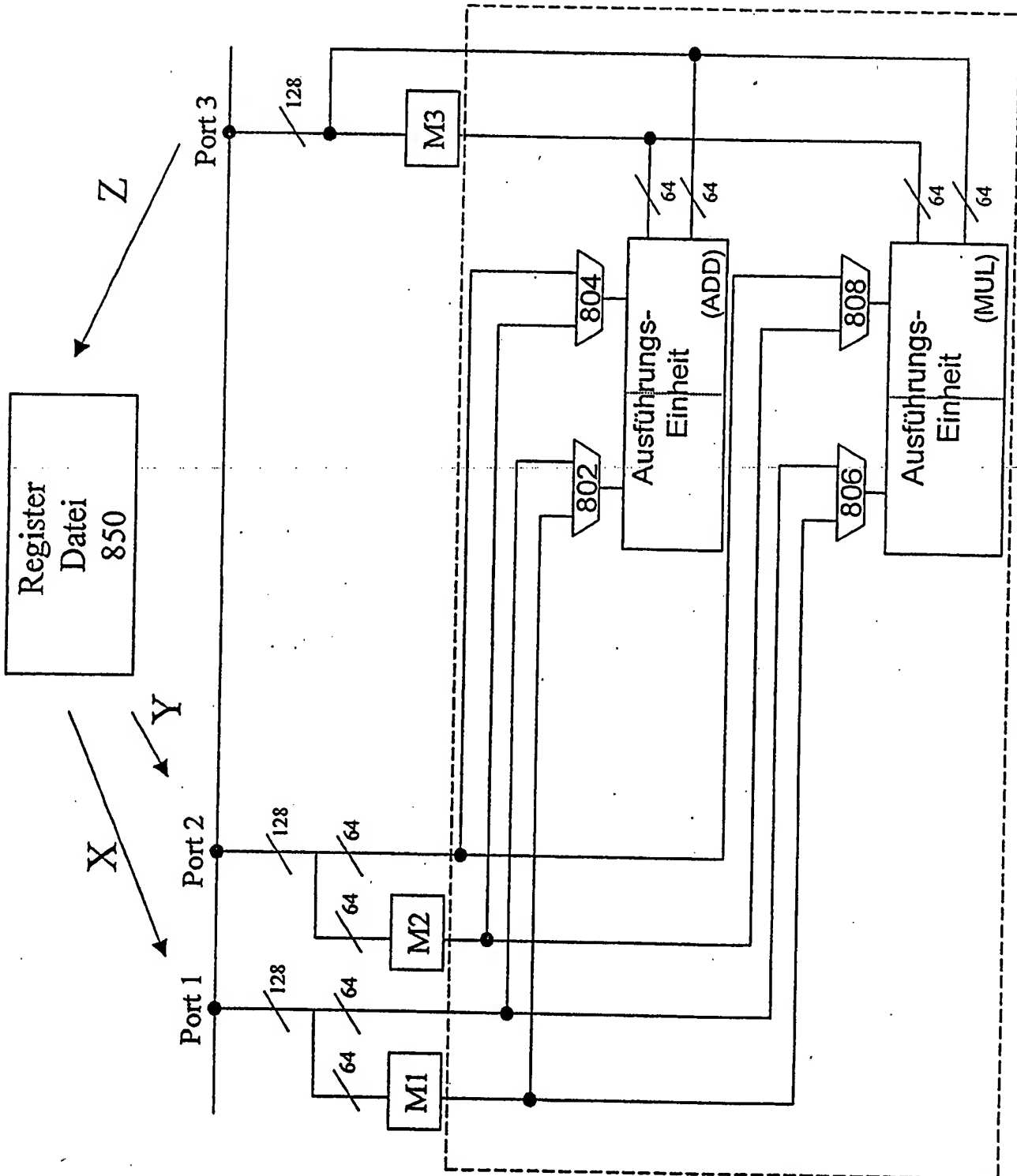
Figur 7A



Figur 7B

\* = entsprechendes Datenelement in X oder Y,

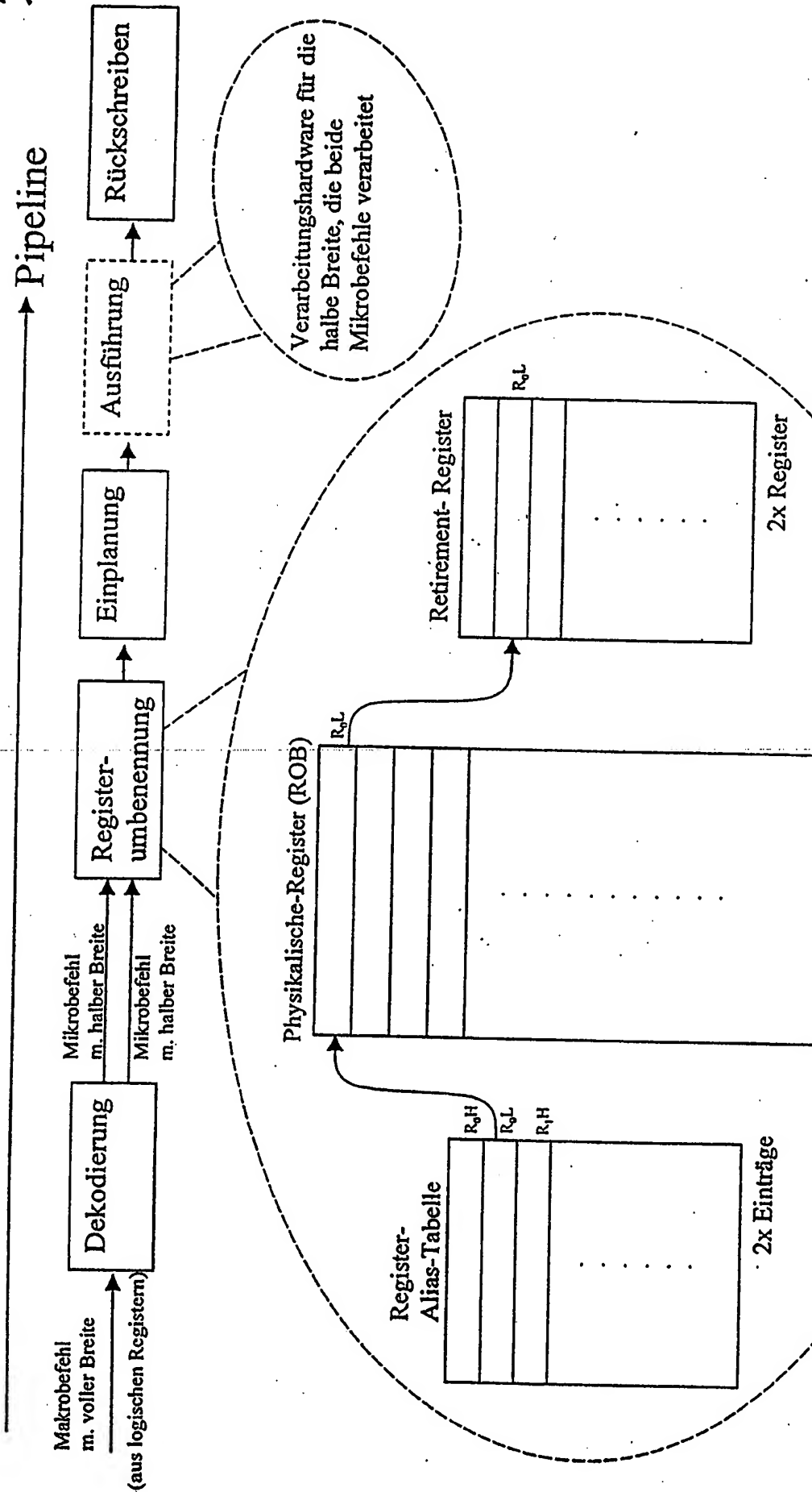
NaN, Ø oder anderer vorgegebener Wert



FIGUR 8A

Zeit	128-Bit-Befehl	durchgeföhrt an 64 Bit Daten
T	ADD X, Y	ADD X0, Y0 ADD X1, Y1
T+1		ADD X2, Y2 ADD X3, Y3
T+1	MUL X, Y	MUL X0, Y0 MUL X1, Y1
T+2		MUL X2, Y2 MUL X3, Y3

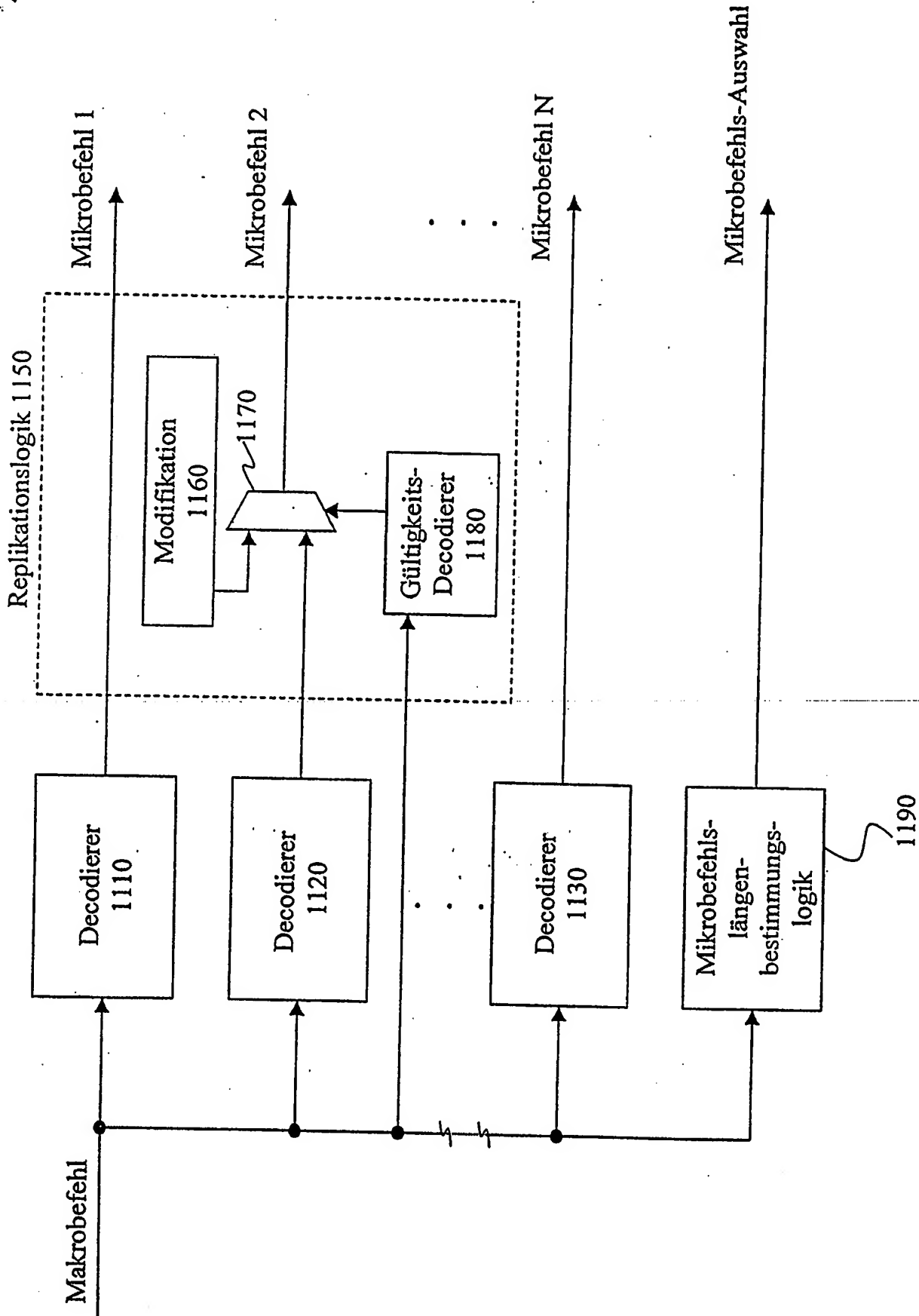
FIGUR 8B



FIGUR 9

Zeit	128-Bit-Befehl	64-Bit-Befehl
T	ADD X, Y	ADD X <sub>L</sub> , Y <sub>L</sub>
T+1		ADD X <sub>H</sub> , Y <sub>H</sub>

FIGUR 10



Figur 11